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

演習の目標

- Xamarin と Xamarin Studio を使用して iOS アプリケーションを Xamarin ネイティブの手法と 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 Mac OS X、iOS SDK、Xcode、Xamarin 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
- Xamarin.Forms (Portable)プロジェクトの作成、開発
- iOS 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、Xamarin 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 Mac OS X、iOS SDK、Xcode、Xamarin の設定を確認

1.1　Xamarin を使用して iOS アプリケーションを開発するには以下が必要です。

- Mac OS X Yosemite(10.10)以上の Mac OS X
- 最新の Xcode
- 最新の iOS SDK
- 最新の Xamarin

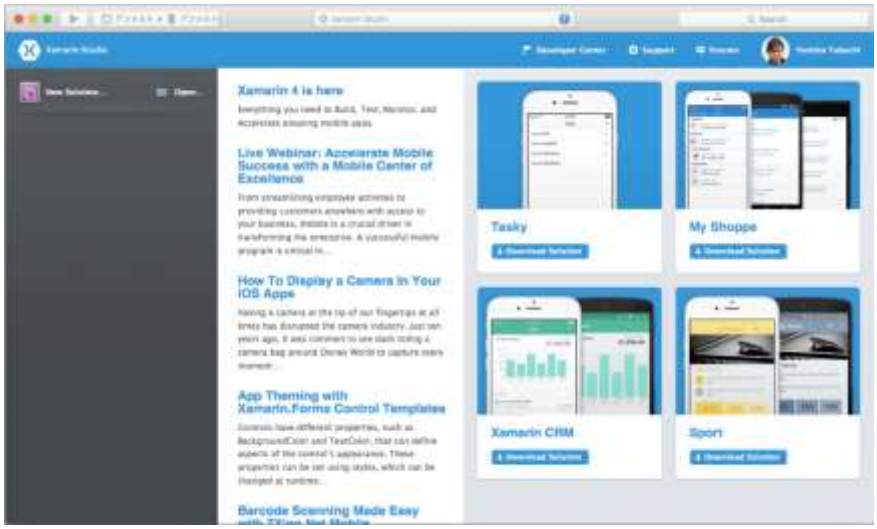
Mac OS X、Xcode、iOS SDK が最新になっていることを確認したら、「Mac に Xamarin.iOS をインストール」（http://www.xlsoft.com/jp/products/xamarin/xamarin_ios_mac_installation.html）を参考に Xamarin をインストールします。

1.2　iOS Simulator または実機の準備

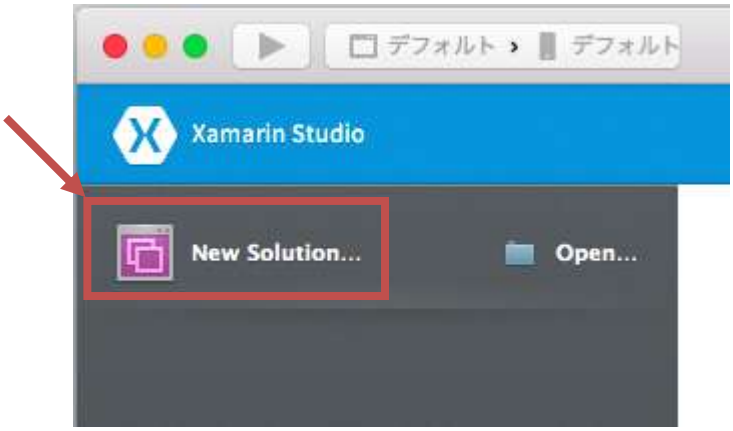
作成したアプリをデプロイする iOS Simulator または実機を用意します。実機へのデプロイは Apple Developer Program（https://developer.apple.com/programs/jp/）に加入するか、Free Provisioning を利用する方法があります。Xamarin での Free Provisioning の使用方法は「Xcode 7 と Xamarin Studio Starter で 1 円も払わずに自作 iOS アプリを実機確認する」（http://ytabuchi.hatenablog.com/entry/2015/09/18/191258）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

2　Hello.iOS 入門

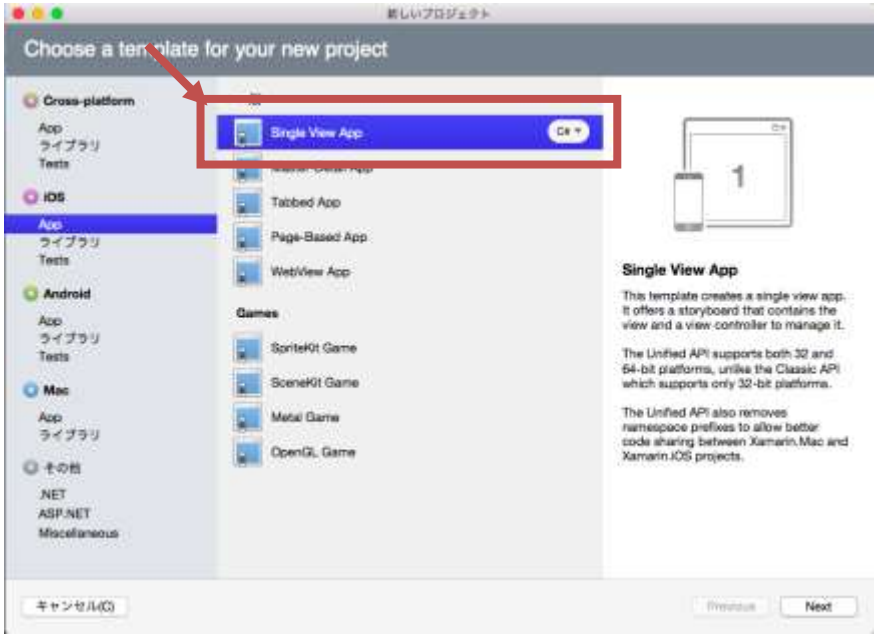
2.1　[アプリケーション]フォルダまたは[Spotlight]から Xamarin Studio を起動して、起動画面を開きます。



2.2　左上の[New Solution...]をクリックして、新規に Xamarin.iOS ソリューションを作成します。

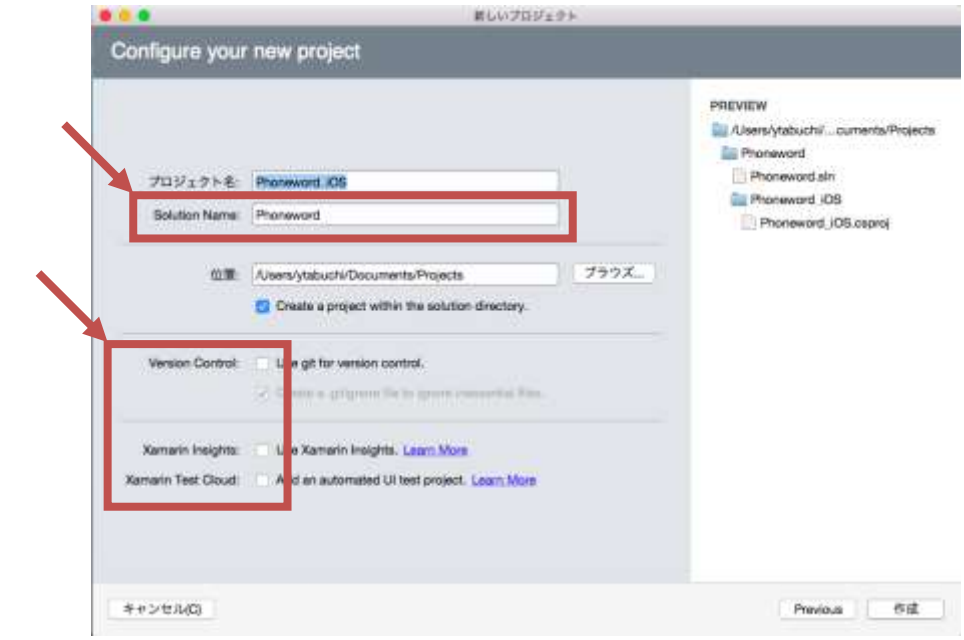


2.3　[新しいプロジェクト]ダイアログから、[iOS > App > Single View App]のテンプレートを選択し、[Next]をクリックします。

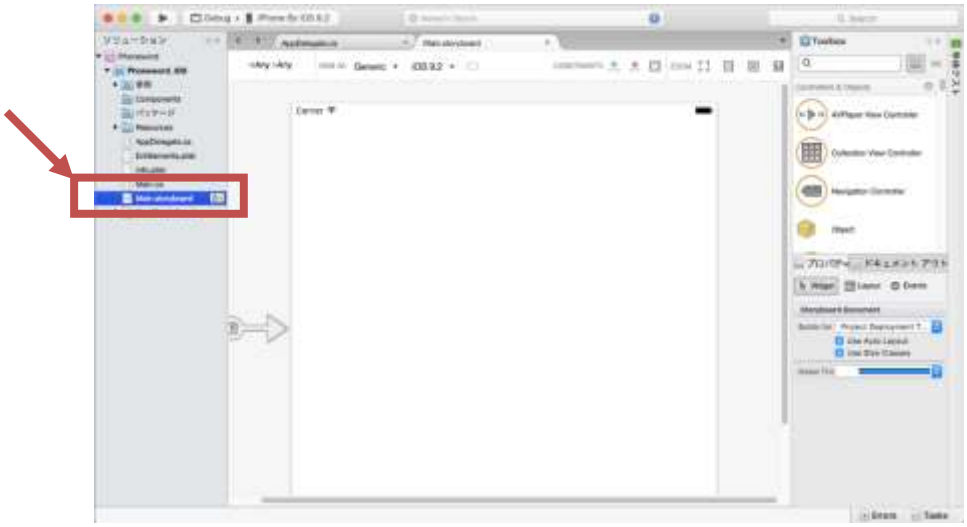


2.4　[App Name]に「Phoneword_iOS」と入力して[Next]をクリックします。

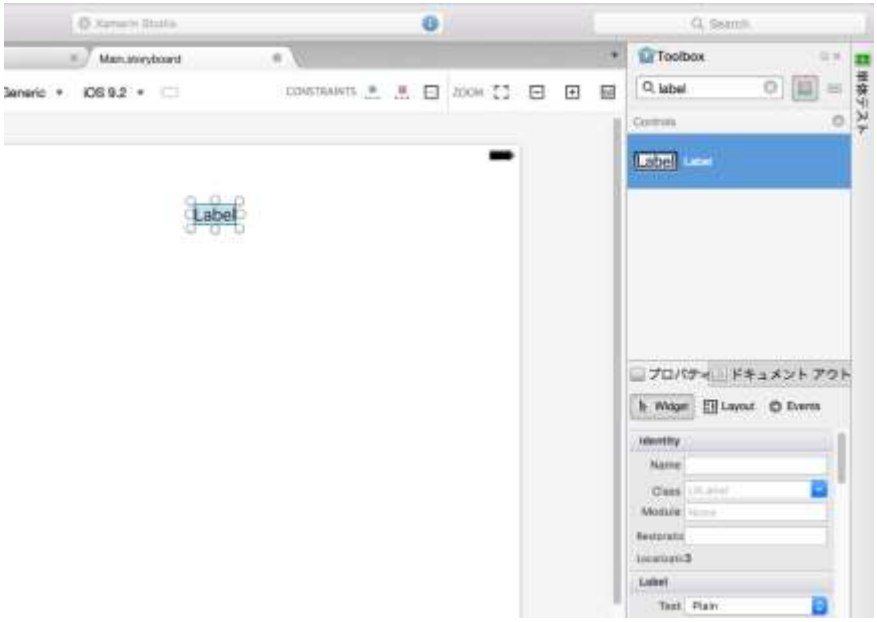
2.5　[Solution Name]を「Phoneword」に変更して[Version Control]、[Xamarin Insights]、[Xamarin Test Cloud]のチェックが外れていることを確認して[作成]をクリックします。



2.6　[ソリューション]パッドの「Main.storyboard」をダブルクリックしてファイルを開きます。

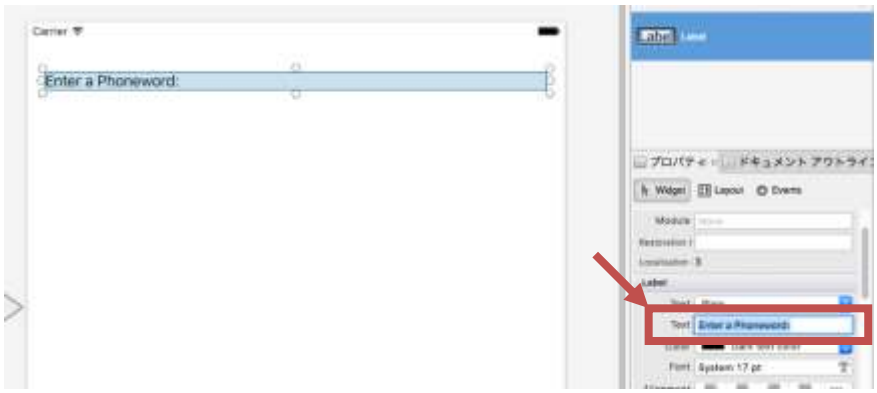


2.7　[Toolbox]（画面右側）にある検索バーで「label」と入力し、デザイン画面（画面中央）に[Label]をドラッグします。

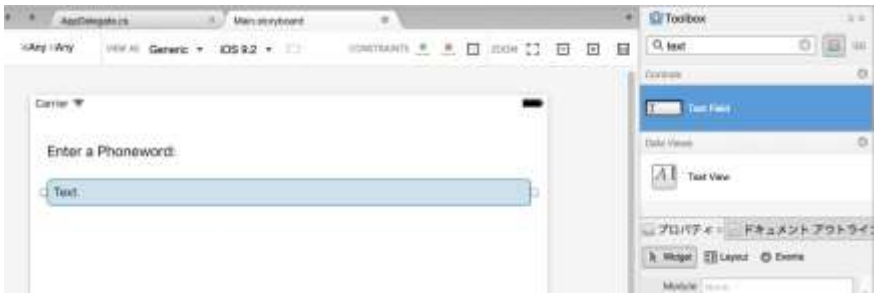


2.8　[Dragging Controls] (コントロールの囲いの○) のハンドルをつかみ、ラベルを広げます。

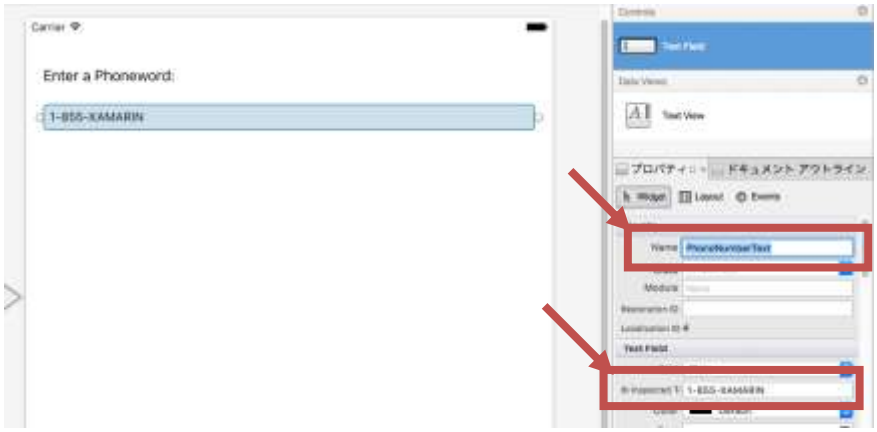
2.9　デザイン画面で選択した[Label]で、[プロパティ]パッドを使用して、[Label]の[Text]プロパティを「Enter a Phoneword:」に変更します。



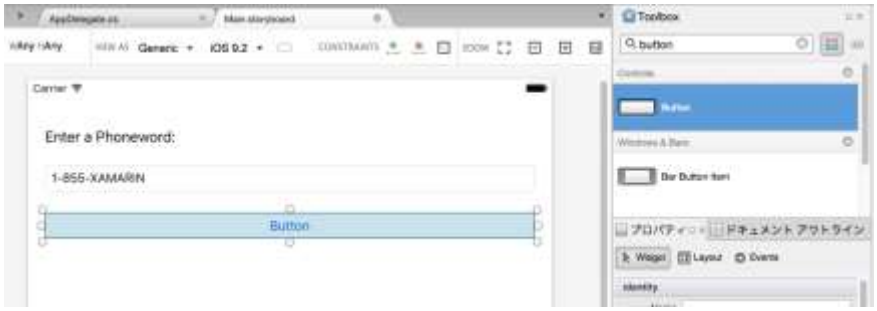
2.10　次に、[Toolbox]で「text field」を検索し、デザイン画面で[Toolbox]から[Text Field]をドラッグし、[Label]の下に[Text Field]を配置します。[Text Field]と[Label]の幅が同じになるように調整します。



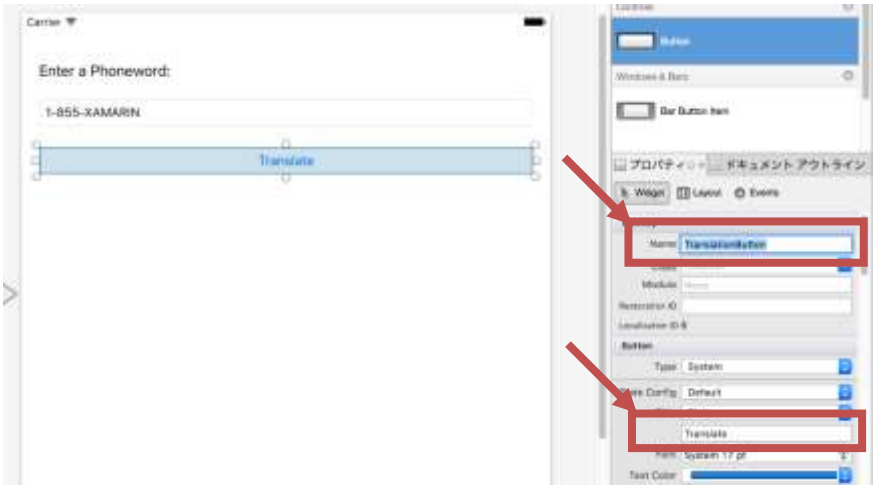
2.11　デザイン画面で選択した[Text Field]で、[プロパティ]パッドの[Widget]の[Identity]セクションの[Name]プロパティを「PhoneNumberText」に変更し、[Ib Inspected Text]プロパティを「1-855-XAMARIN」に変更します。



2.12　次に、[Toolbox]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。

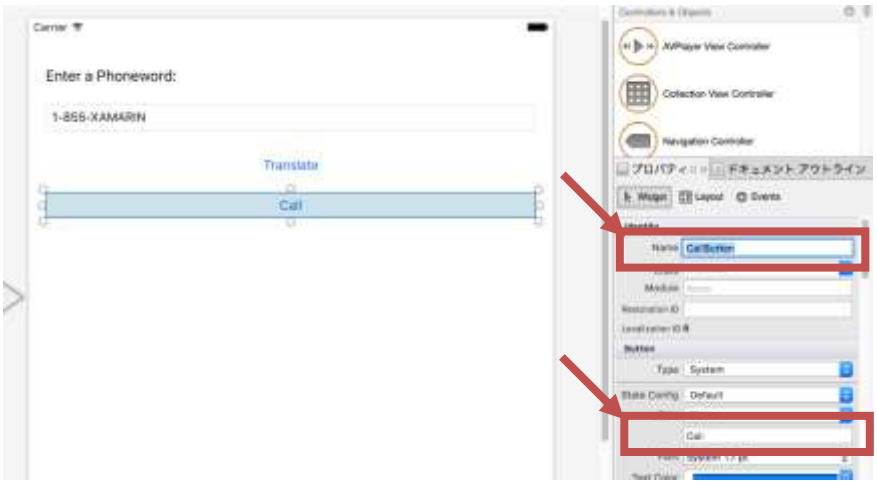


2.13　デザイン画面で選択した[Button]で、[プロパティ]パッドの[Identity]セクションの[Name]プロパティを「TranslateButton」に変更します。[Title]プロパティを「Translate」に変更します。



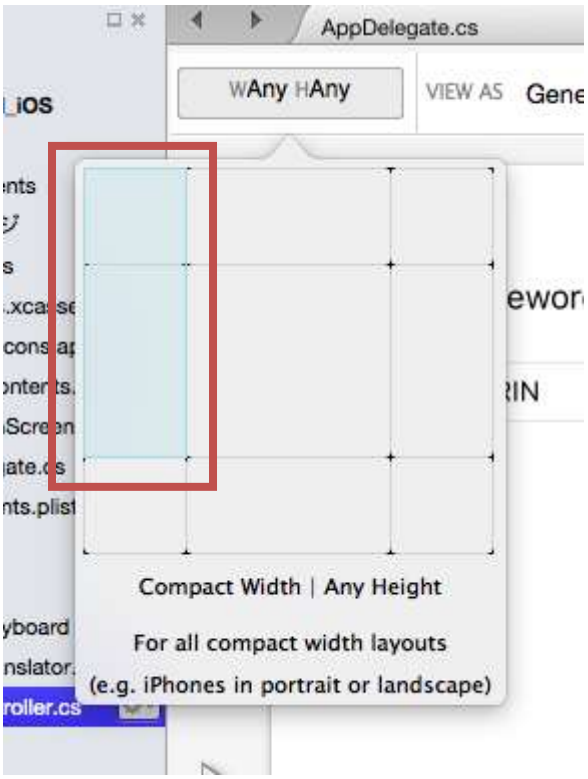
2.14　上記 2 つのステップを繰り返し、デザイン画面の[Toolbox]から[Button]をドラッグし、最初の[Button]の下に配置します。その[Button]の幅を最初の[Button]と同じ幅になるように調整します。

2.15　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。

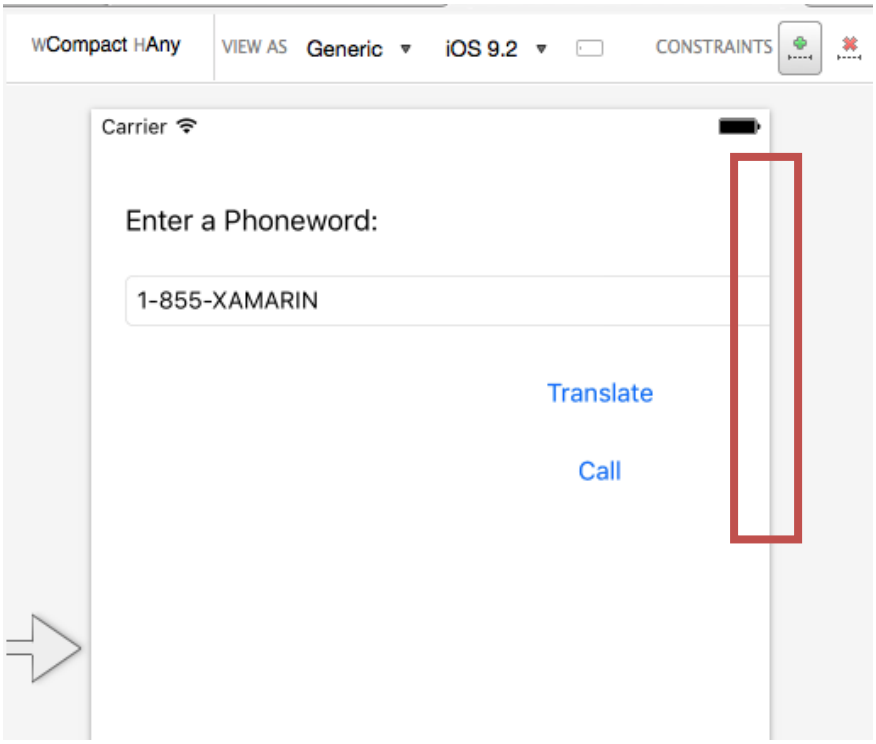


2.16　[ファイル ＞ 保存]または[⌘ + S]ボタンを押して作業内容を保存します。

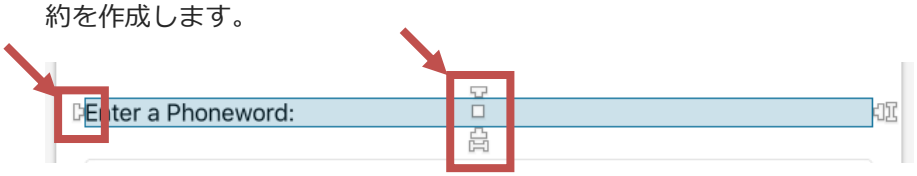
2.17　デザイン画面の左上の[Size Class]ボタンから[Compact Width ＞ Any Height]をタップします。



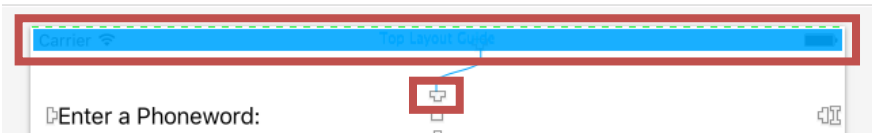
2.18　[Compact Width > Any Height]は iPhone 5/6/6Plus などで使用されるレイアウトですが、作成したオブジェクトが隠れてしまっているのが分かります。



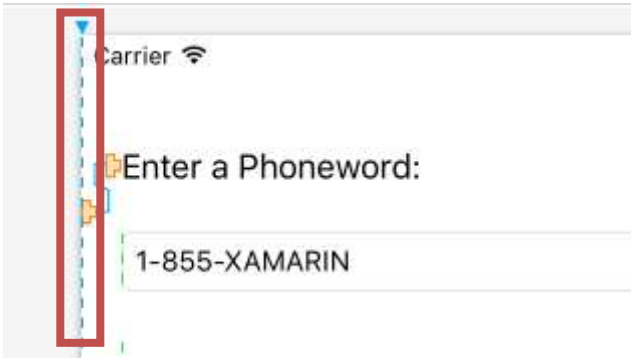
2.19　[Any > Any]のレイアウトに戻し、[Constraints]（制約）を追加することですべてのレイアウトに対応できるようにします。[Label]を選択して再度クリックすると、ハンドラーが[○]から[□]や[凸]などに変わります。これらのハンドラーを操作して、制約を作成します。



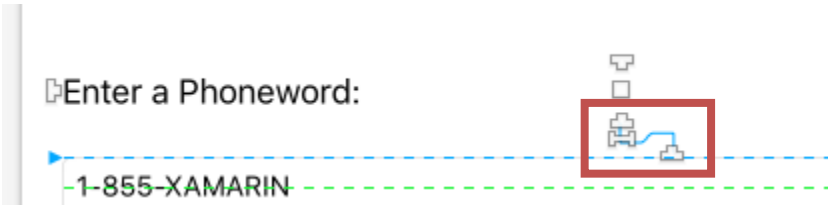
2.20　[Label]の上部のハンドラーをドラッグして、[View]の上部でドロップします。



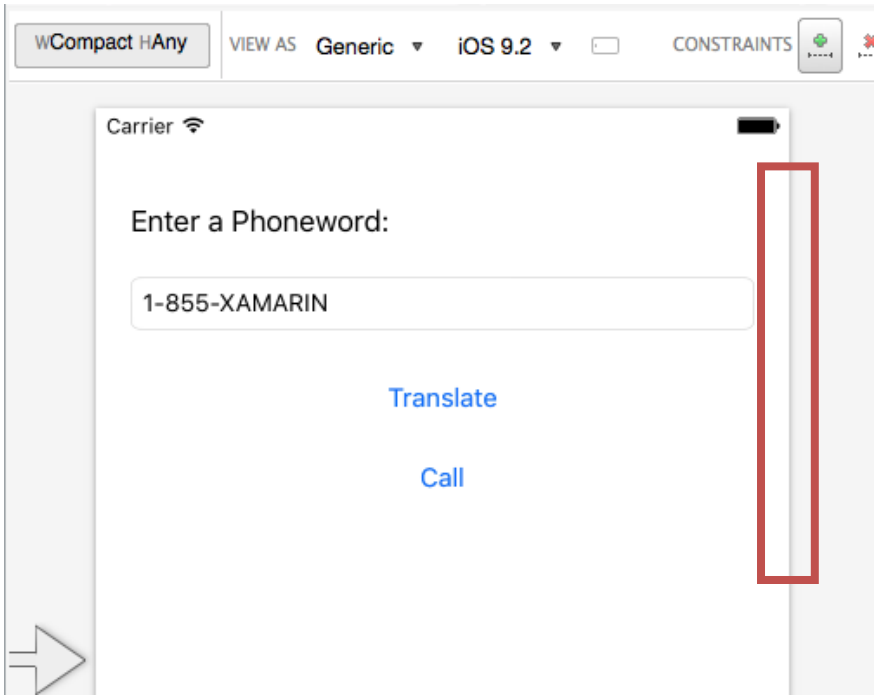
2.21　左側のハンドラーをドラッグして、[View]の左端でドロップします。[制約]を指定できる個所になると、青い点線が表示されるので活用してください。



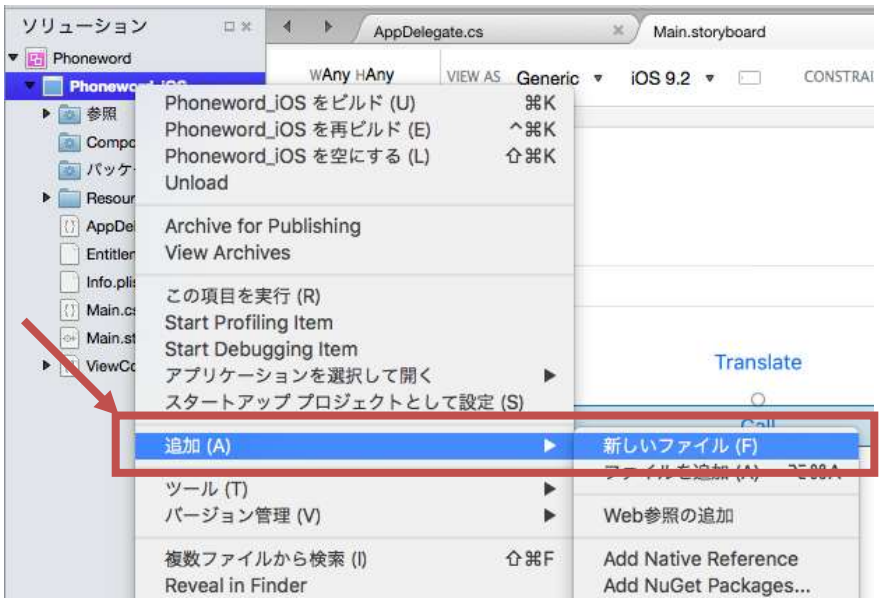
2.22　[Label]と[EditText]など、オブジェクト同士の距離も[制約]を指定します。



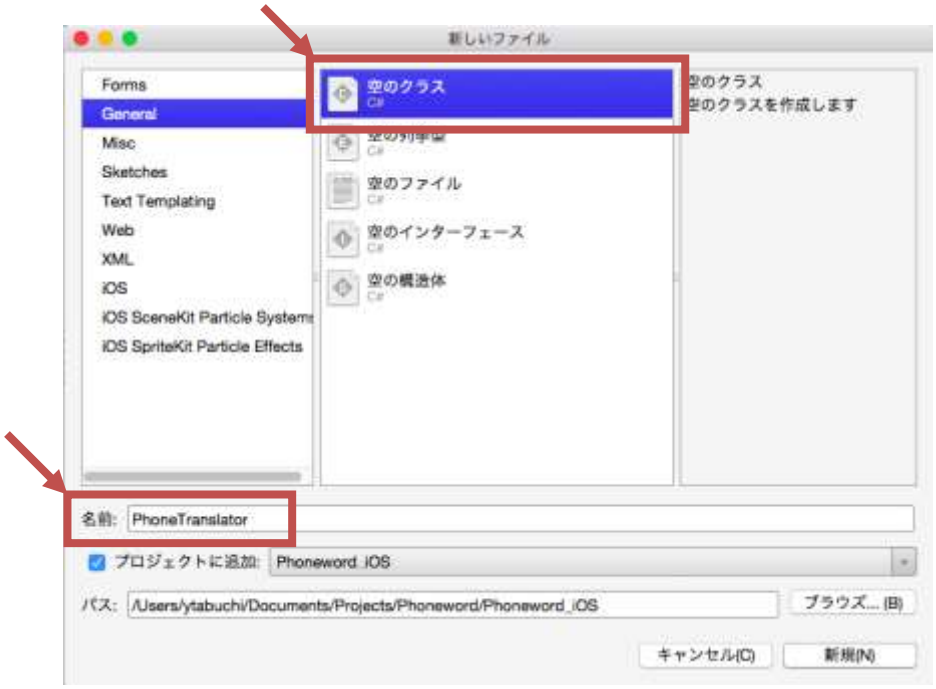
2.23　同様に、すべてのオブジェクトに[制約]を指定してください。すべて指定すると、[Compact Width > Any Height]を表示しても各オブジェクトが画面内に収まるのが分かります。



2.24　次に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]パッドの[Phoneword_iOS]プロジェクトを右クリックして[追加 > 新しいファイル]を選択または[⌘ + N]ボタンを押します。



2.25　[新しいファイル]ダイアログで、[General > 空のクラス]を選択し、新しいファイルに「PhoneTranslator」と名前を付けます。



2.26　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;
            return null;
        }
    }
}
```

[PhoneTranslator.cs]ファイルを保存して閉じます。

2.27 次にコードを追加して、[ViewController]クラスの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]パッドの[ViewController.cs]をダブルクリックして、開きます。

2.28　[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
}
```

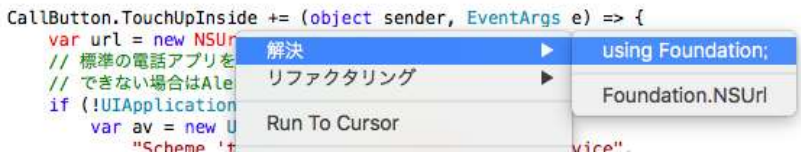
2.29　[TranslateButton]と名前を付けた最初のボタンをユーザーが押した際に応答するコードを追加します。以下のコードを[ViewDidLoad]の最後に追加します。

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

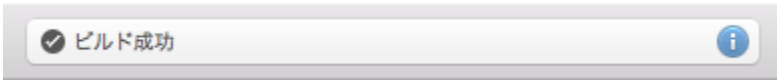
2.30　次に、[CallButton]と名前を付けた二番目のボタンをユーザーが押した際に応答するコードを追加します。[TranslateButton]のコードの下に以下のコードを追加します

```
CallButton.TouchUpInside += (object sender, EventArgs e) => {
  var url = new NSUrl ("tel:" + 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
    // できない場合は AlertView 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]上で右クリックし、[解決 > using Foundation;]を選択して[using]を追加します。



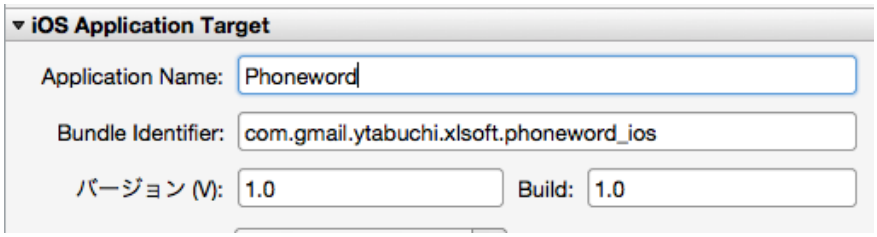
2.31 作業を保存し、[ビルド > すべてビルド]を選択または[⌘ + B]ボタンを押して、アプリケーションをビルドします。[AppIcons.appiconset]のエラーが表示される場合は、エラーメッセージをダブルクリックして再度ビルドすると表示されなくなります。アプリケーションをコンパイルすると、IDE の上位に[ビルド成功]とメッセージが表示されます。



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。

2.32 これで、アプリケーションが動作するので、最後の仕上げを加えていきましょう！[Info.plist]ファイルのアプリケーション名とアイコンを編集します。[ソリューション]パッドの[Info.plist]をダブルクリックして開きます。

2.33 [iOS Application Target]セクションで、[Application Name]を「Phoneword」に変更します。

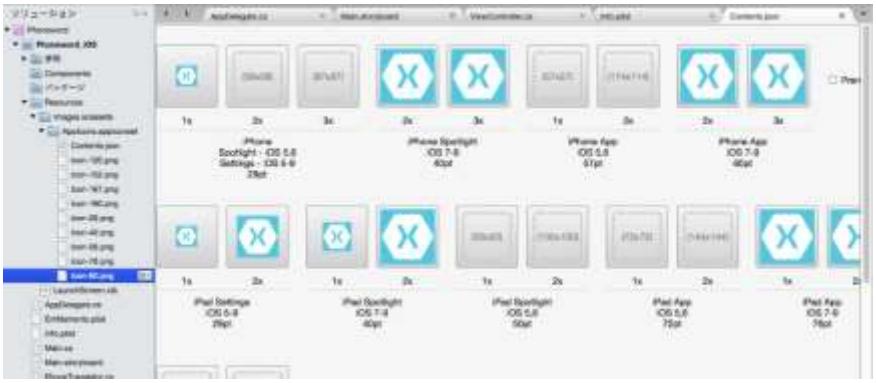


2.34 アプリケーションのアイコンと起動イメージを設定するために、まずアイコンセット
（ https://github.com/ytabuchi/XamarinHOL/blob/master/XamarinAppIcons.zip ）
をダウンロードします。

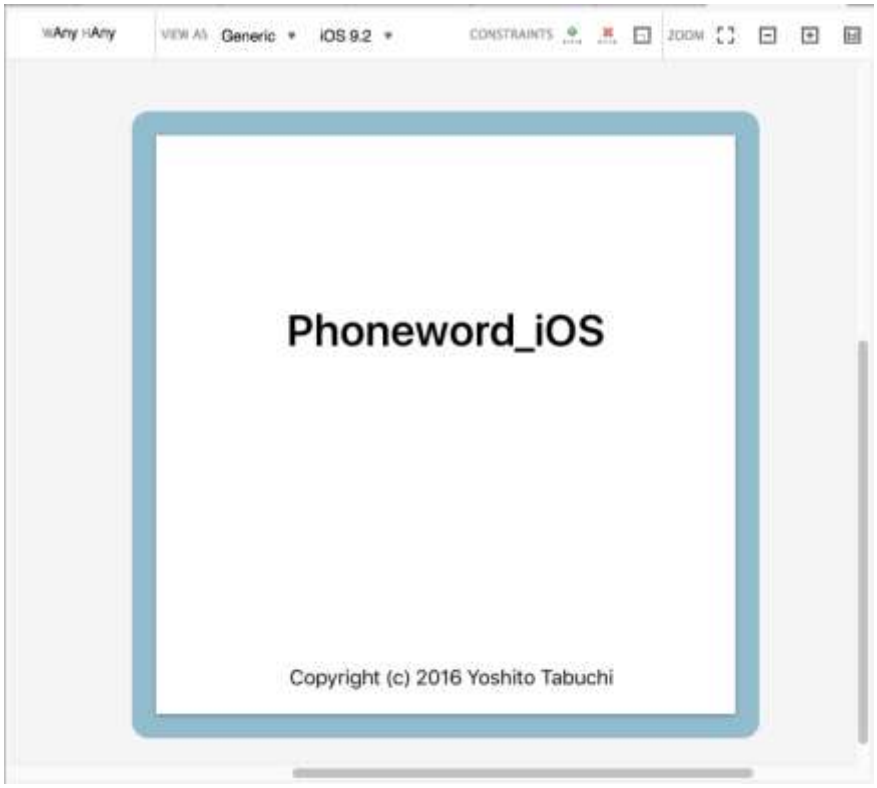
2.35 アプリケーションのアイコンは[Contents.json]ファイルで指定します。[ソリューション]パッドから[Contents.json]ファイルを見つけ、ダブルクリックして開きます。

2.36　ここでは iOS 7 以上に必要なアイコンを追加していきます。

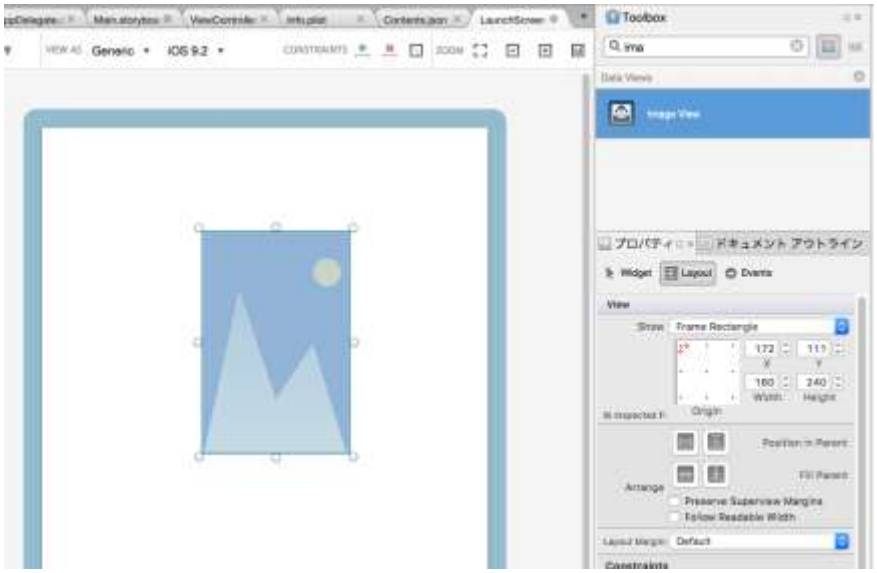
2.37　すべてのファイルを登録すると次のようになります。



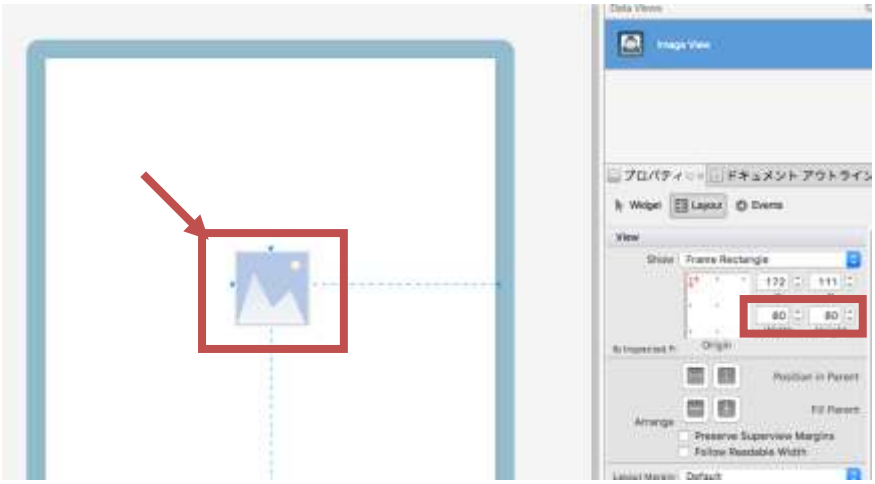
2.38　次に[LaunchScreen]の設定を行います。[LaunchScreen.xib]ファイルをダブルクリックして開きます。



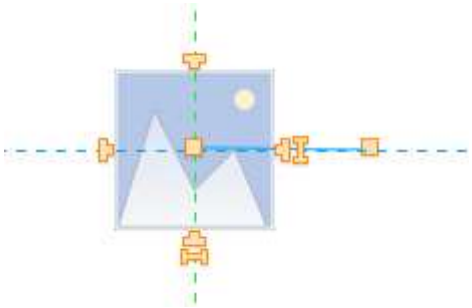
2.39　コントロールはすべて削除し、[Toolbox]から「Image」を入力して検索し、[ImageView]を配置します。



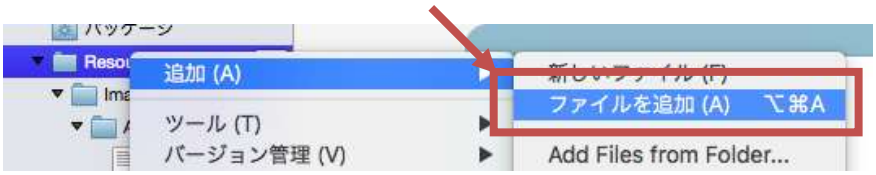
2.40　[プロパティ]パッドの[Layout]タブの[View]セクションの[Width]と[Height]を[80/80]に変更し、View の中央に配置します。



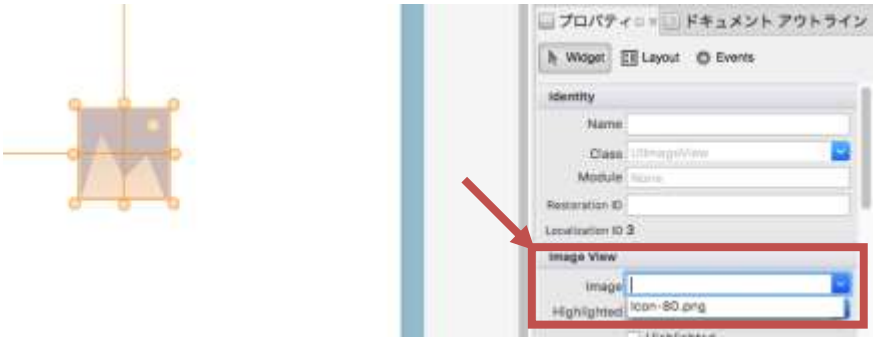
2.41　中央に[Image View]の位置を固定するため、[Image View]の中央の[□]をドラッグして縦横に[制約]を作成します。



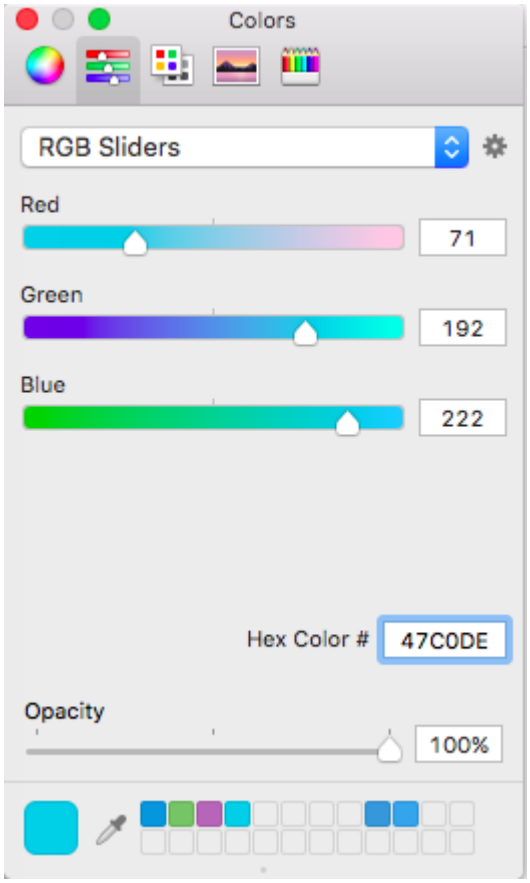
2.42　[Image View]の画像を指定するため、[Resources]フォルダに[Icon-80.png]をコピーします。[Resources]フォルダを右クリックし、[追加 ＞ ファイルを追加]を選択します。



2.43　[Image View]を選択し、[プロパティ]パッドの[Widget]タブの[Image View]セクションの[Image]で追加した[Icon-80.png]を選択します。



2.44　[View]の背景を同じ色にするため、[View]を選択し、[プロパティ]パッドの[Widget]タブの[View]セクションの[Background]をクリックし、[Other]で[カラーパネル]を表示し、[Hex Color #]に「47C0DE」を指定します。



2.45　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。IDE の左上で、最初のドロップダウンから[Debug]を選択し、2 番目のドロップダウンから[iPhone 6s iOS 9.2]を選択し、Start ボタン[▶]を押します。

2.46　これで、iOS Simulator または実機でアプリケーションが起動します。（画面は Simulator）

2.47　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。

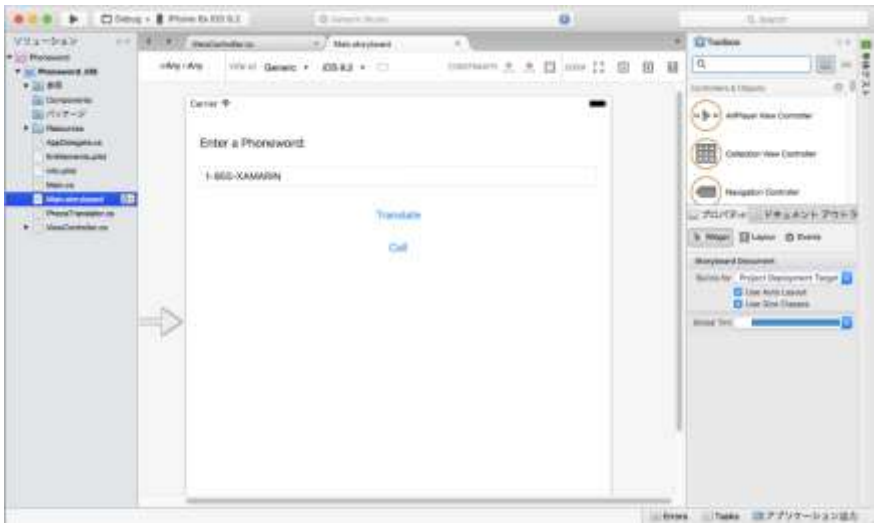
初めての Xamarin.iOS アプリケーションの完成です！次のステップ Hello, iOS Multiscreen Quickstart で、このガイドで習得したツールとスキルをさらに試しましょう。

3　Hello, iOS Multiscreen Quickstart

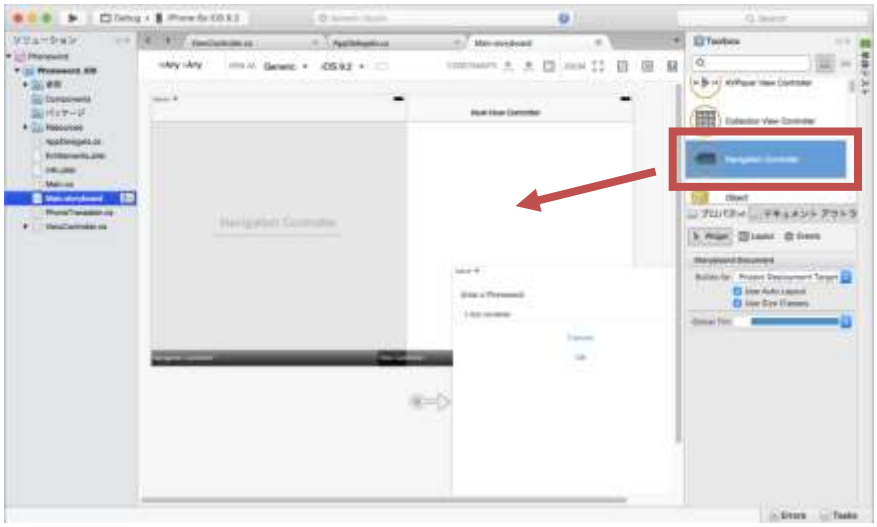
このセクションでは、Phoneword アプリケーションにもう一つ画面を追加し、その画面にこのアプリの通話履歴を残す方法を説明します。本ガイドで完成したアプリケーションでは、以下のスクリーン ショットのように、2 番目の画面に通話履歴を表示します。

3.1　Xamarin Studio 上で Phoneword プロジェクトを開きます。

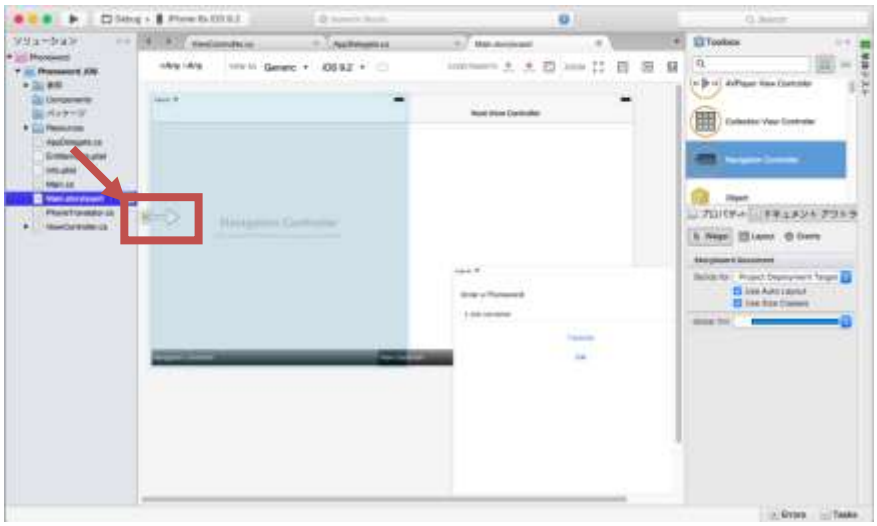
3.2　まずはユーザーインターフェースの編集から始めます。[ソリューション]パッドから[Main.storyboard]ファイルを開きます。



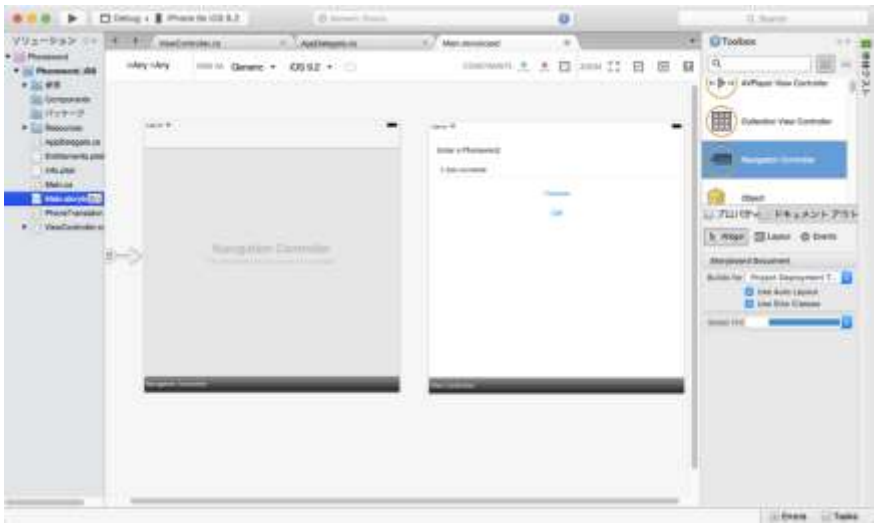
3.3　ツールボックスから、デザイン画面に Navigation Controller をドラッグします



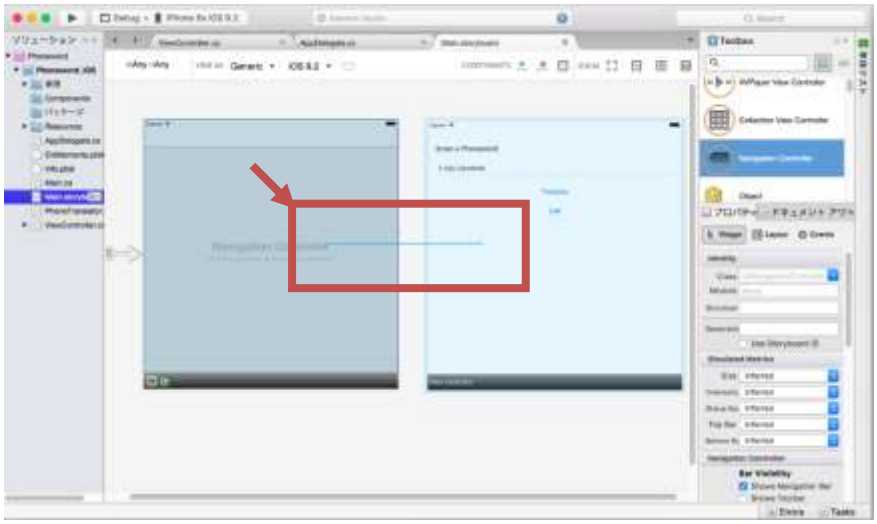
3.4　[Sourceless Segue]（Phoneword 画面の左にある灰色の矢印）を[Phoneword]画面から[Navigation Controller]へドラッグし、アプリケーションのスタートポイントを変更します。



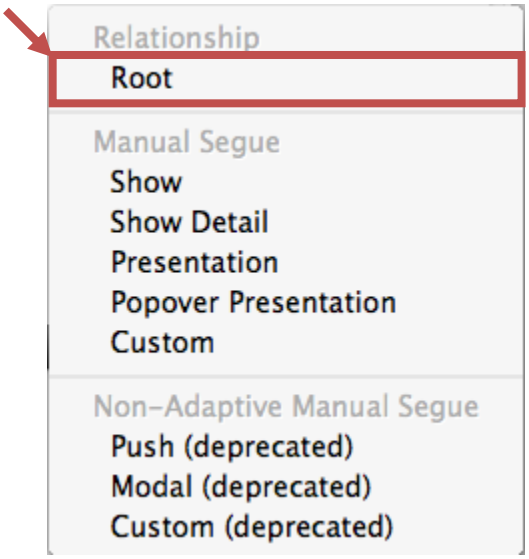
3.5　[Root View Controller]下部の黒いバーをクリックして選択し、[Delete]キーを押してデザイン画面から削除します。その後、[Navigation Controller]を[Phoneword]画面の横に移動させます



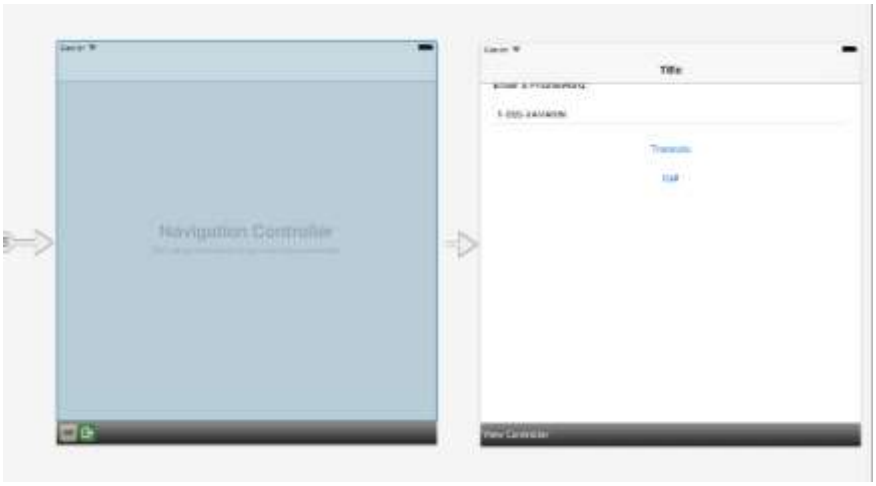
3.6　[Navigation Controller]の[Root View Controller]として[ViewController]をセットします。[Ctrl]キーを押し、[Navigation Controller]内部をクリックします。青い線が表示されるので、[Ctrl]キーを押したまま[Navigation Controller]から[Phoneword]画面までドラッグします。



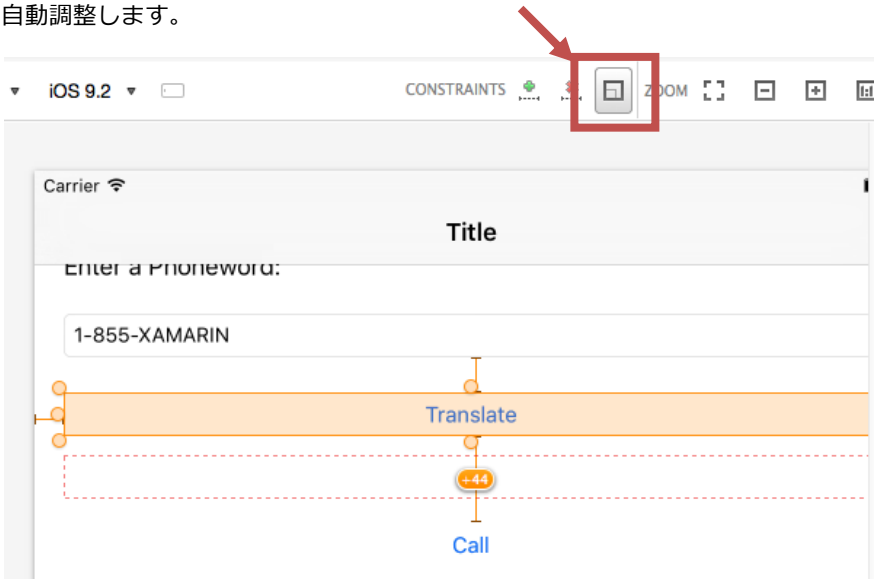
3.7　ポップオーバーから、[Relationship]の[Root]をクリックします。



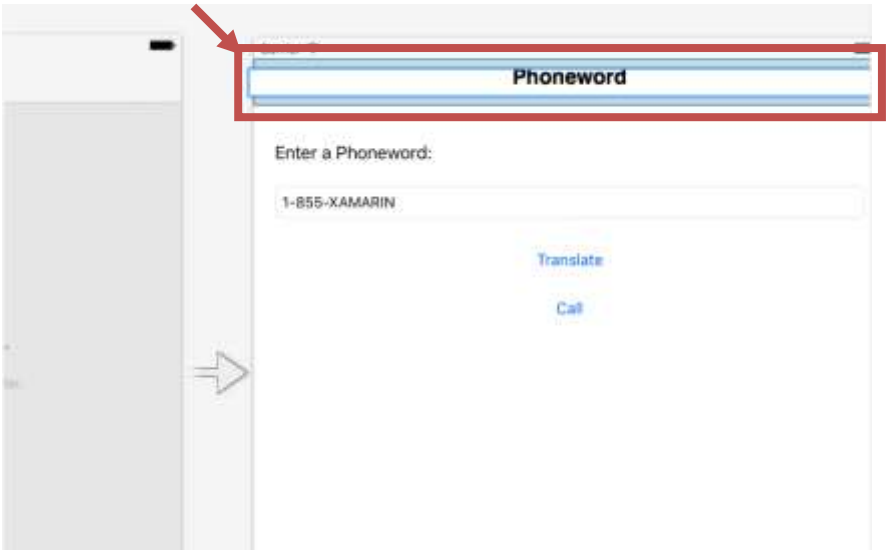
3.8　これで[ViewController]が[Navigation Controller]の[Root View Controller]になりました。



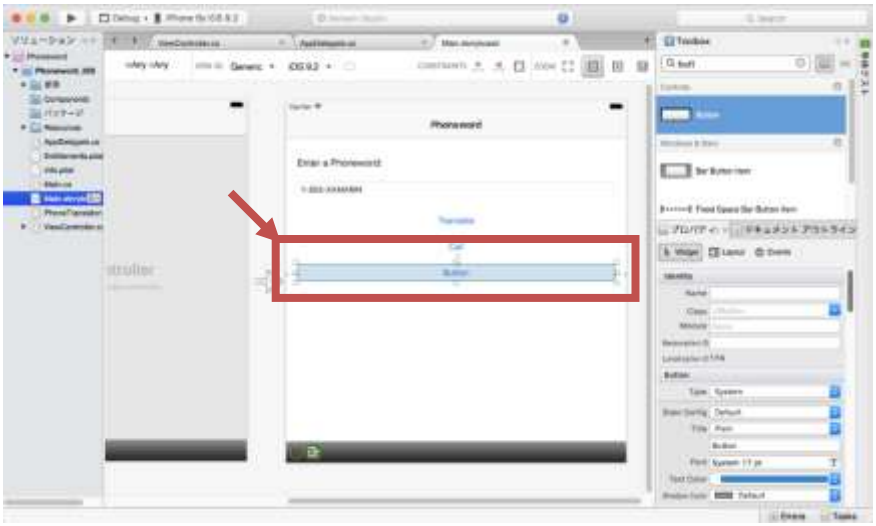
上記のようにオブジェクトが隠れてしまう場合は、[制約]が[Navigation Controller]に合わせて下がってきているため、[Update frames based on constraints]ボタンでオブジェクトの場所を自動調整します。



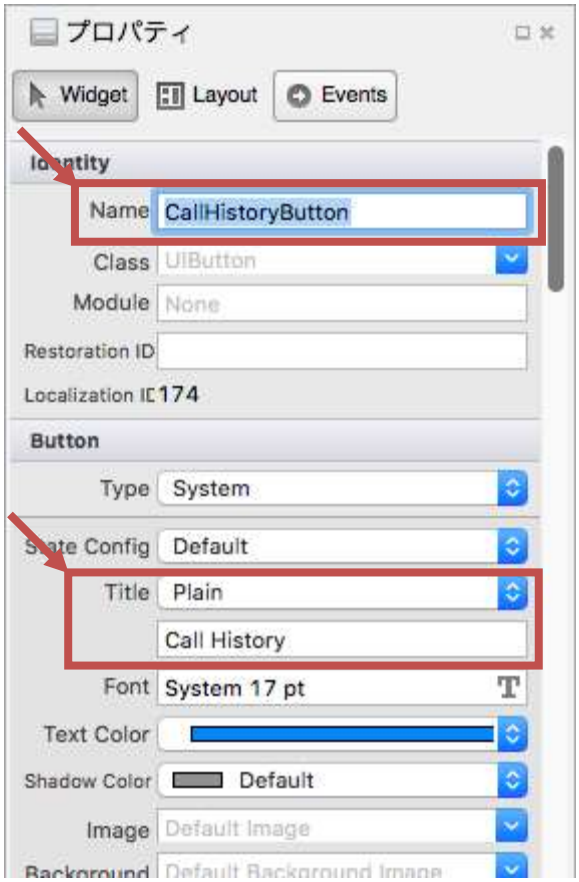
3.9　[Phoneword]画面のタイトルバーをダブルクリックし、[Title]の文字列を「Phoneword」に変更します。



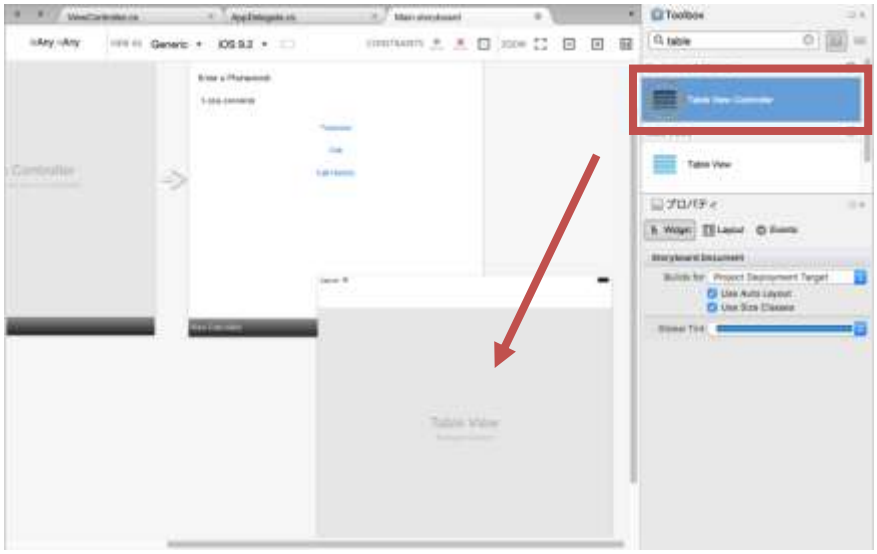
3.10　ツールボックスから[Button]をドラッグし、[Call]ボタンの下に配置します。ハンドルをドラッグして、新しいボタンを[Call]ボタンと同じ幅に合わせます。



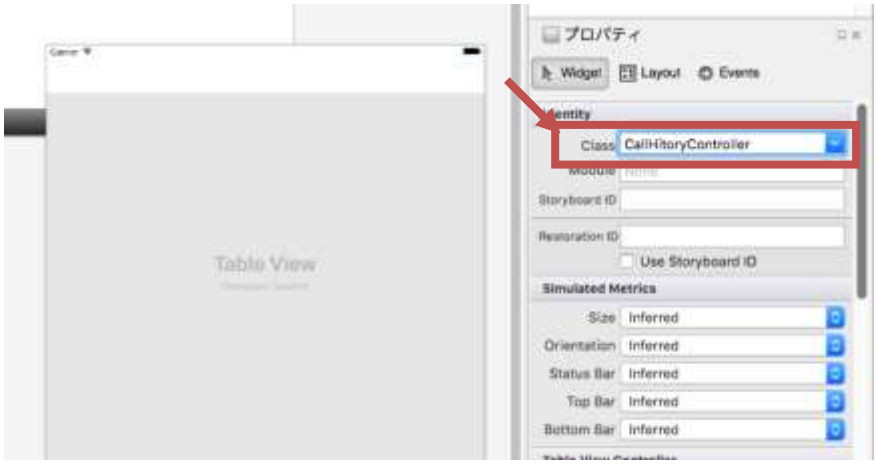
3.11　[プロパティ]パッドで、ボタンの名前を[CallHistoryButton]に変更し、タイトルを「Call History」に変更します。



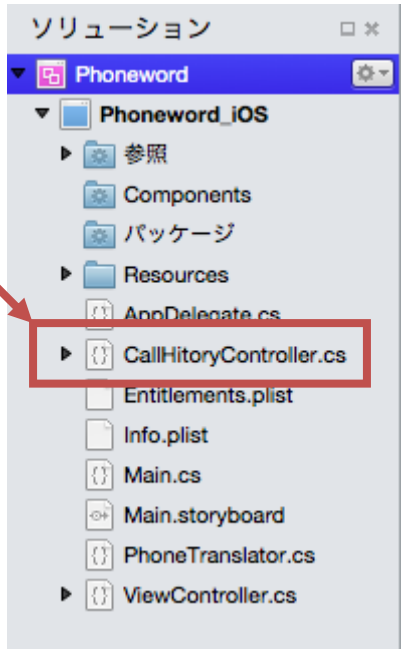
3.12　[Call History]画面を作成します。[Toolbox]からデザイン画面に[Table View Controller]をドラッグします。



3.13　[Table View Controller]の下の黒いバーをクリックして[Table View Controller]を選択します。[プロパティ]パッドで[Table View Controller]のクラスを「CallHistoryController」に変更し、[Enter]キーを押します。



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 パッドに表示されます。



3.14　[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。

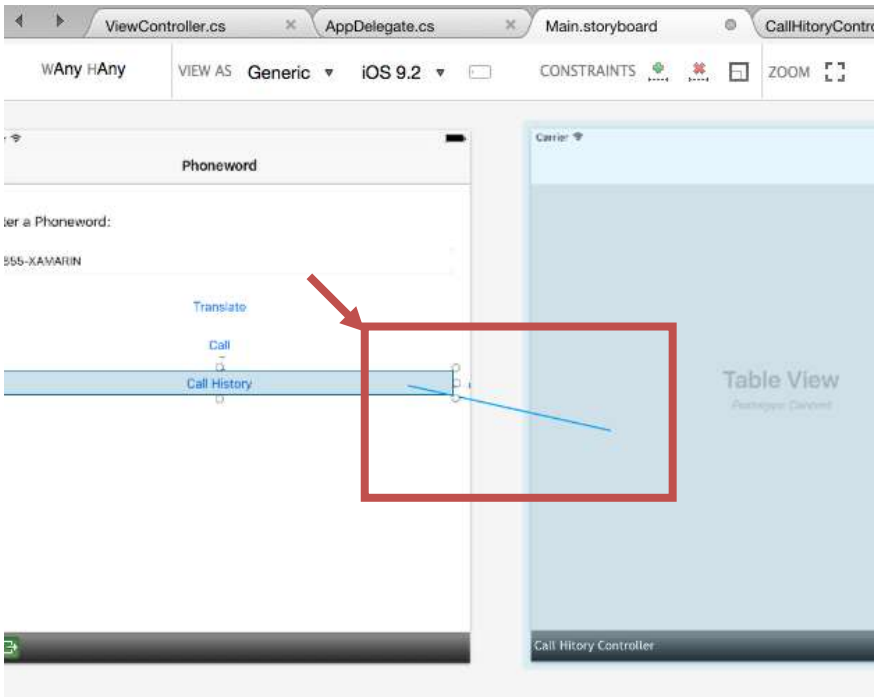
```
using System;
using Foundation;
using UIKit;
using System.Collections.Generic;

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;
            }
        }
    }
}
```

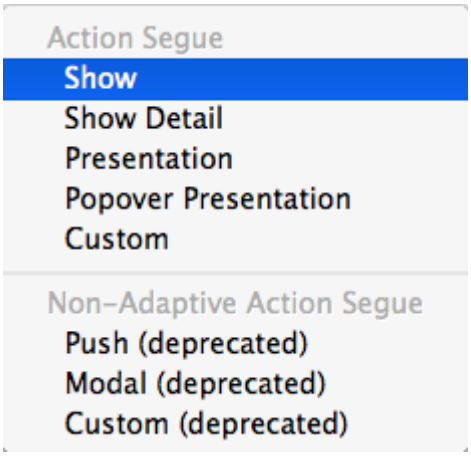
アプリケーションを保存し、エラーがないかビルドを実行して確認します。

3.15 [Phoneword]画面と[Call History]画面の切り替え（Segue）を作成します。

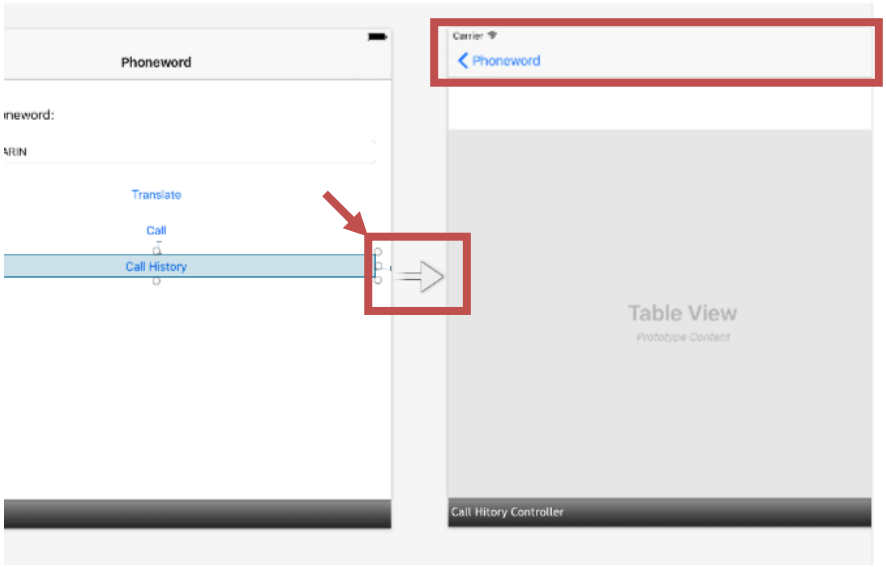
[Phoneword]画面で[Call History]ボタンを選択し、ボタンから[Call History]画面に[Ctrl]を押しながらドラッグします。



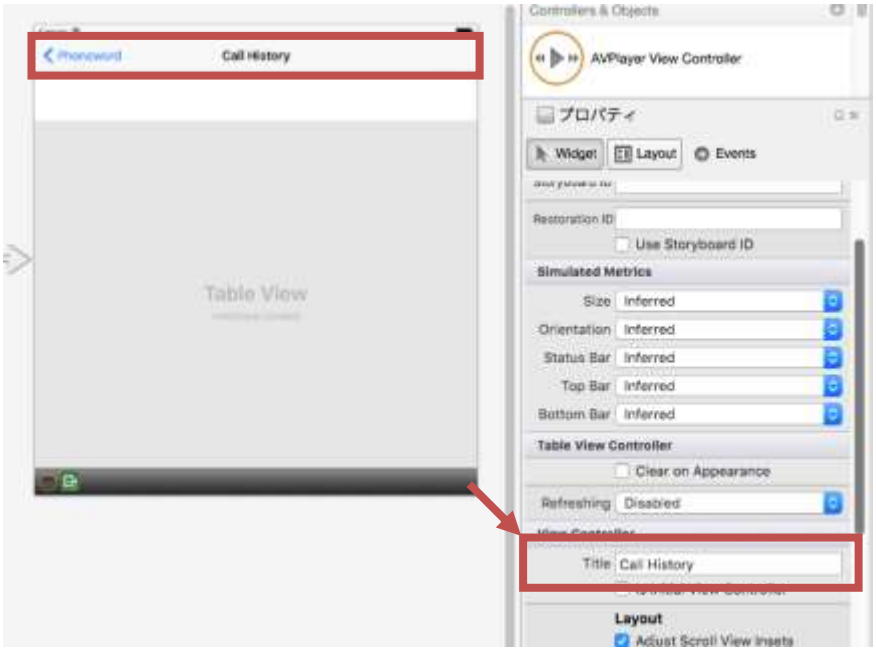
3.16　[Action Segue]のポップオーバーから[Show]を選択します。



3.17　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。



3.18　画面下の黒いバーを選択して[Table View Controller]にタイトルを追加します。[プロパティ]パッドの[View Controller]の[Title]を「Call History」に変更します。



3.19　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。

まずは、電話を掛けた番号を保存するコードが必要です。電話を掛けた[PhoneNumbers]を文字列のリストとして保存していきます。
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;
```

3.20　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...
    }
}
```

3.21 [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。

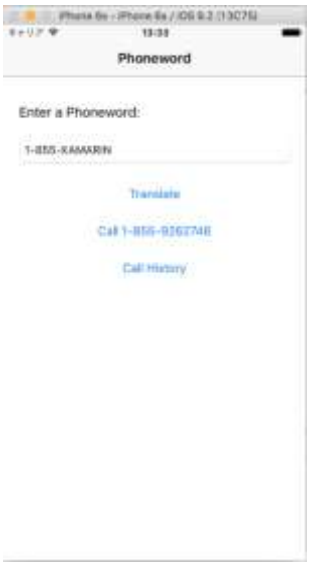
```
CallButton.TouchUpInside += (object sender, EventArgs e) => {
                // 変換した電話番号を PhoneNumbers に追加します
                PhoneNumbers.Add (translatedNumber);
                // 標準の電話アプリを呼び出すために tel:のプリフィックスで URL ハンドラーを使用します
                var url = new NSUrl ("tel:" + translatedNumber);
                // できない場合は AlertView 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);
                }
            };
```

3.22 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 callHistoryContoller = segue.DestinationViewController as CallHistoryController;
    //set the Table View Controller’s list of phone numbers to the
    // list of dialed phone numbers
    if (callHistoryContoller != null)
    {
        callHistoryContoller.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。

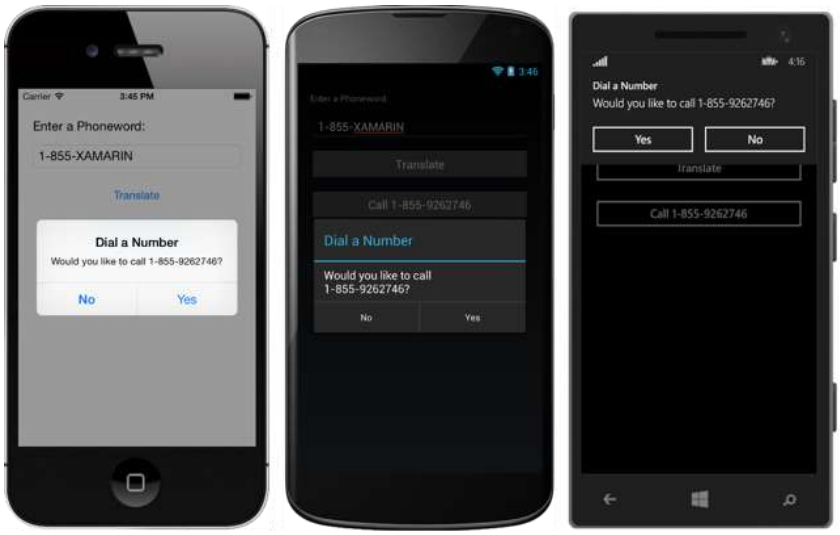
3.23　Start ボタンを押して iOS Simulator 上でアプリケーションを実行します。



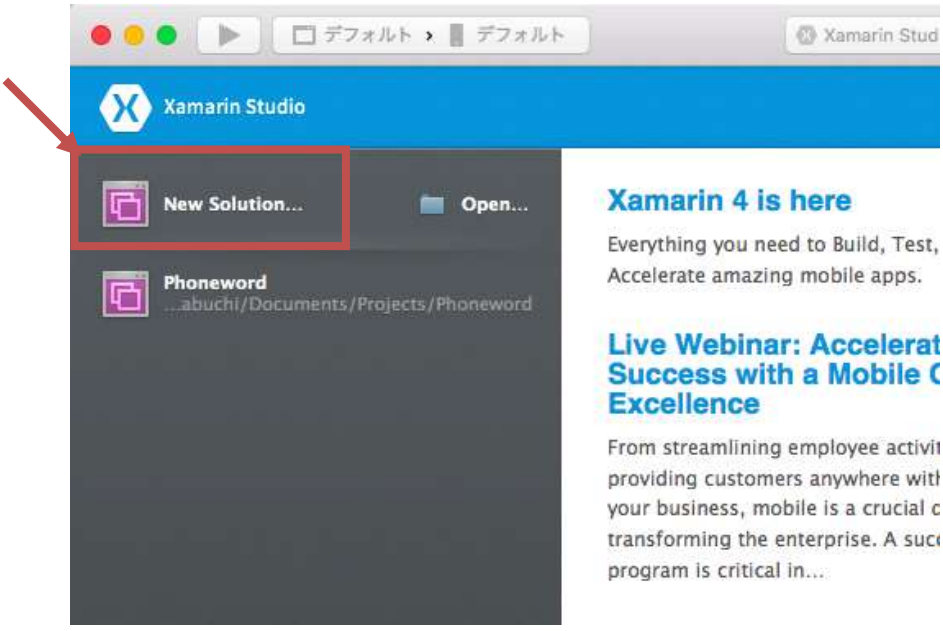
おめでとうございます。複数画面を操作する最初の Xamarin.iOS アプリケーションが完成しました！

4　Xamarin.Forms Quickstart

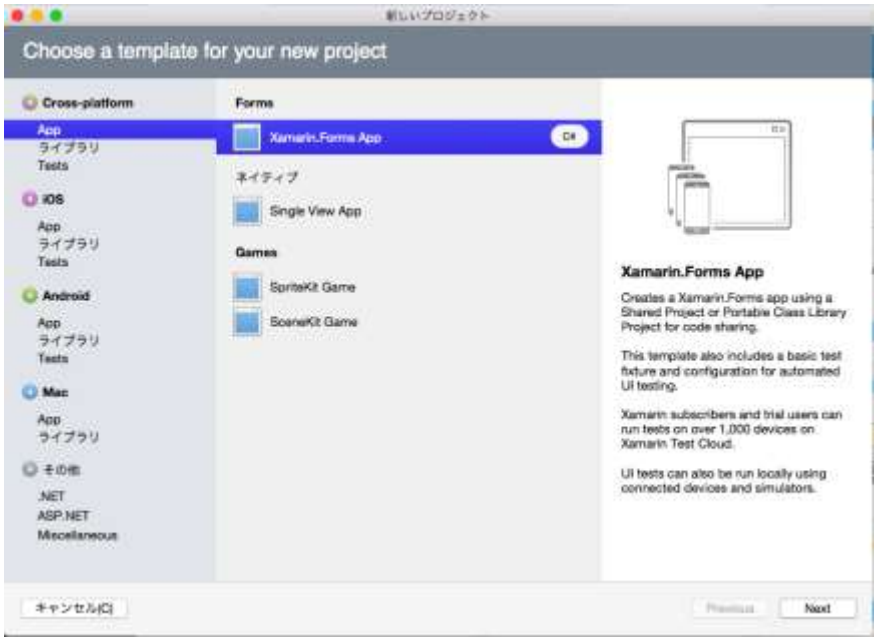
このセクションでは、先ほど Xamarin.iOS プロジェクトで作成した Phone Number アプリを Xamarin.Forms を使用して作成する方法を説明します。アプリケーションの完成図は以下のようになります。



4.1　Xamarin Studio を起動し、[New Solution]をクリックして、新しいソリューションを作成します。

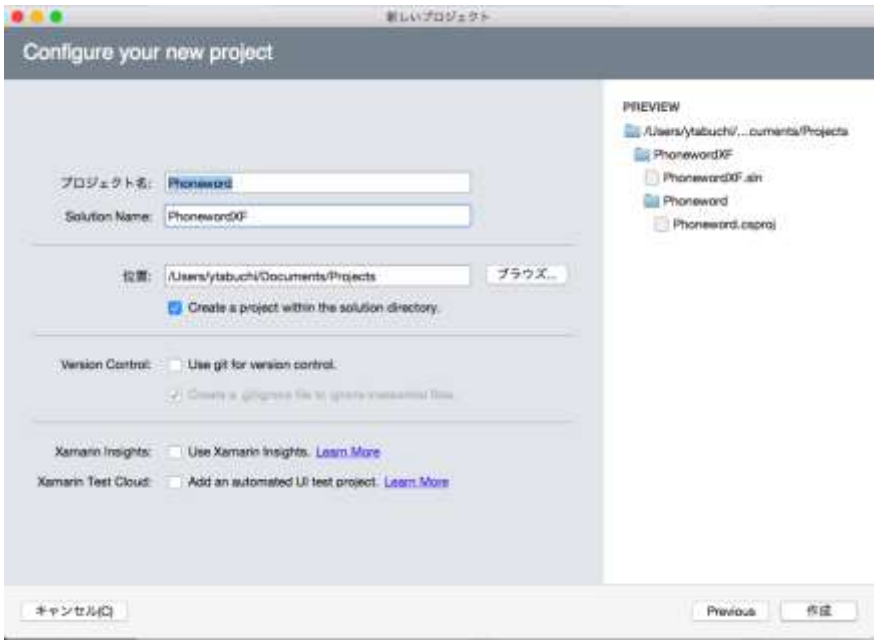


4.2 [新しいプロジェクト]画面で、[Cross-platform > App > Xamarin.Forms App]をクリックします。

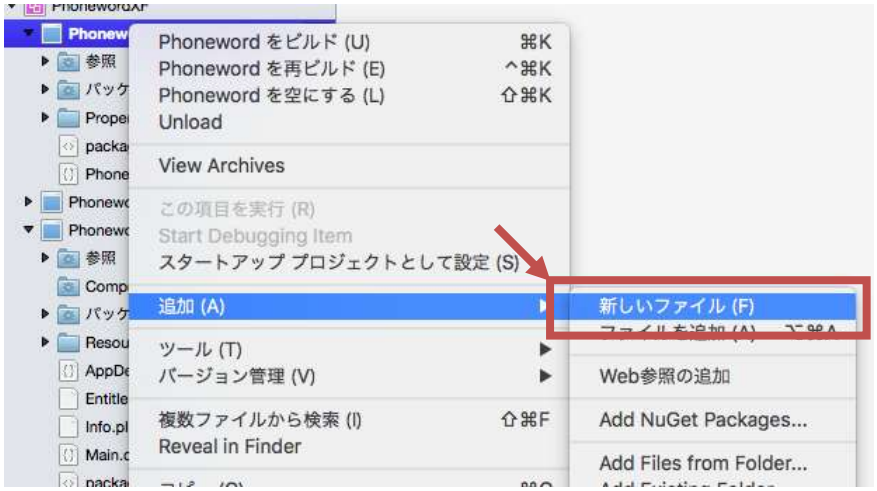


4.3 [App Name]に「Phoneword」と入力し、[Shared Code]が[Use Portable Class Library]が選択されていることを確認し、[Next]をクリックします。

4.4 [Solution Name]を「PhonewordXF」に変更し、[Xamarin Insights]と[Xamarin Test Cloud]のチェックが外れていることを確認し、[作成]をクリックします。（先ほどのXamarin ネイティブのプロジェクト名とかぶってしまうため）

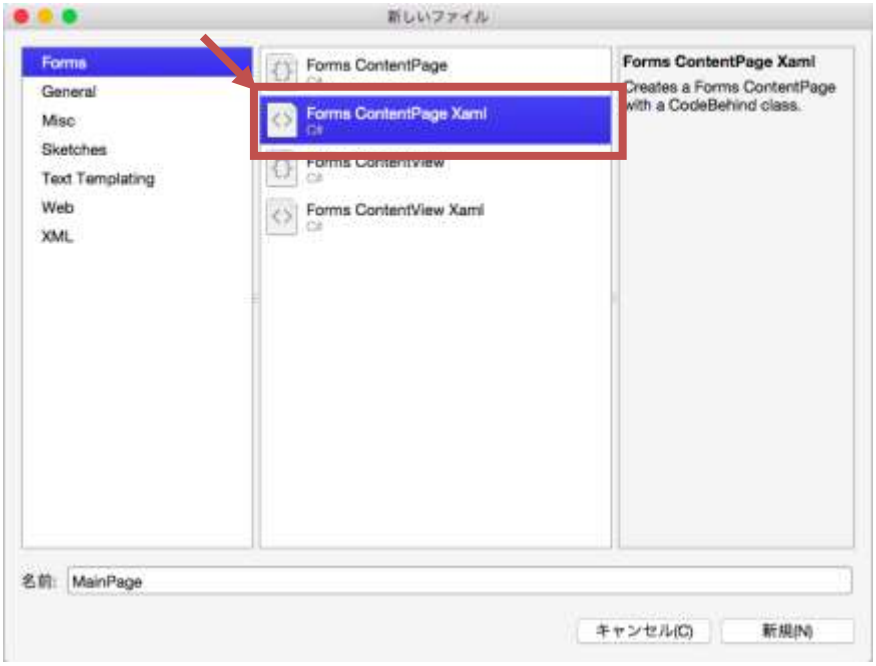


4.5 [ソリューション]パッドで[Phoneword]プロジェクトを右クリックし、[追加 > 新しいファイル]をクリックします。



<< 新しい項目 >>

4.6　[新しいファイル]画面から、[Forms > Forms ContentPage Xaml]を選択し、名前を「MainPage」と付け、[新規]ボタンをクリックします。

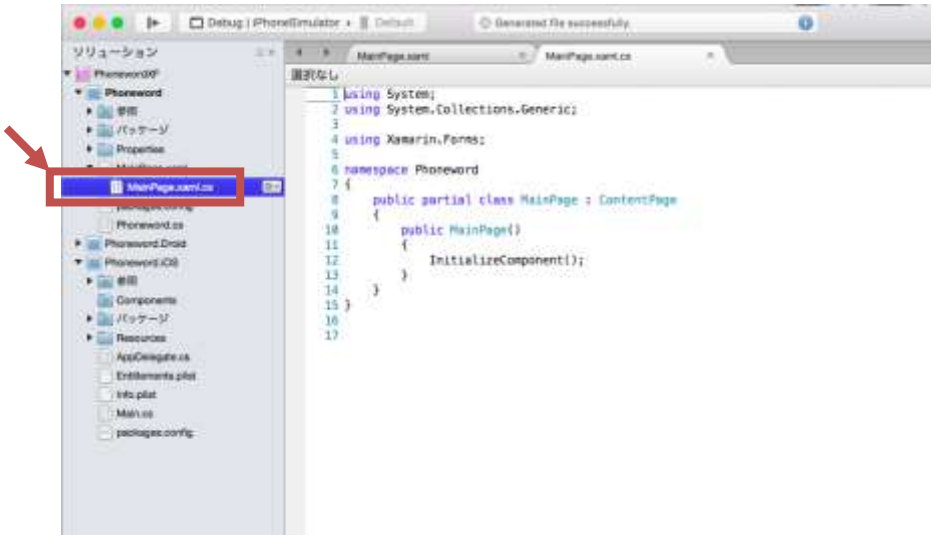


4.7　[MainPage.xaml]ですべてのテンプレートコードを削除し、以下のコードで置き換えます。

```
<?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="translateButton" Text="Translate" Clicked="OnTranslate" />
      <Button x:Name="callButton" Text="Call" IsEnabled="false" Clicked="OnCall" />
    </StackLayout>
  </ContentPage.Content>
</ContentPage>
```

[⌘＋S]を押して変更を保存します。

4.8　[ソリューション]パッドで[MainPage.xaml]を展開し、[MainPage.xaml.cs]をダブルクリックして開きます。



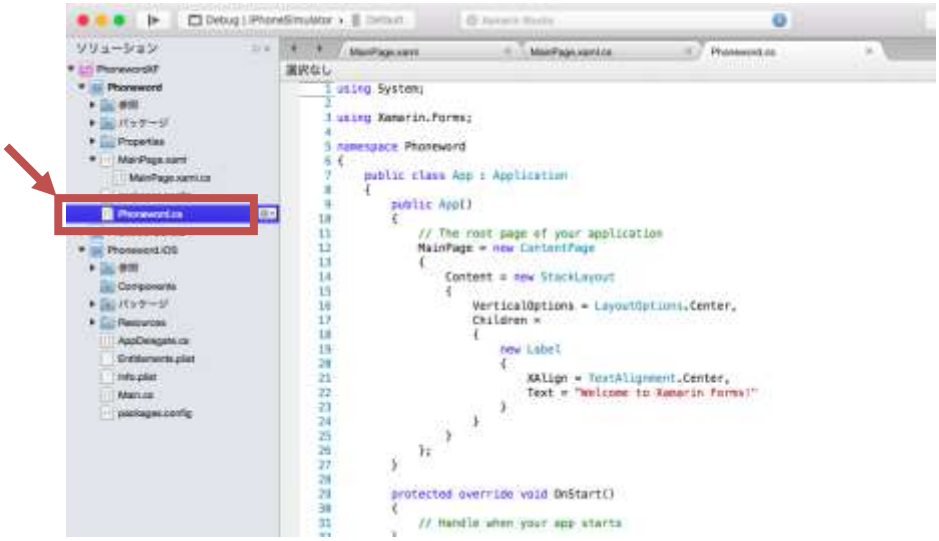
4.9　[MainPage.xaml.cs]ですべてのテンプレートコードを以下のコードで置き換えます。[OnTranslate]と[OnCall]メソッドはユーザーインターフェースの[Translate]と[Call]ボタンがクリックされた時にそれぞれ実行されます。

```
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);
            }
        }
    }
}
```

[⌘＋S]を押して変更を保存します。この時点でいくつか参照できないクラス、メソッドがありますが、以降の手順で作成していきます。

4.10　[ソリューション]パッドで[Phoneword.cs]をダブルクリックして開きます。



4.11　[Phoneword.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。

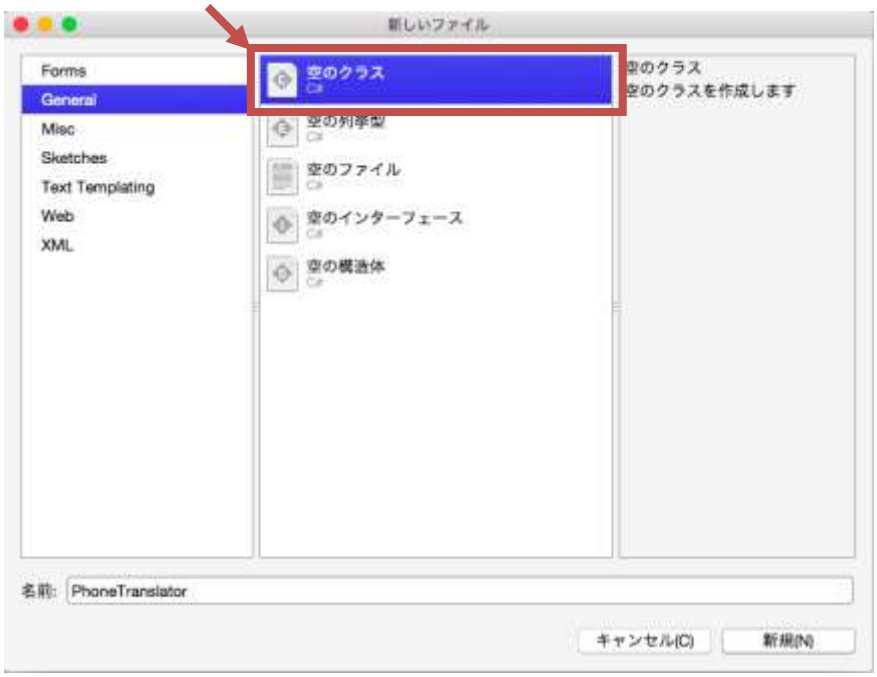
```
using System;
using Xamarin.Forms;

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
        }
    }
}
```

[⌘＋S]を押して変更を保存します。

4.12 [ソリューション]パッドで、[Phoneword]プロジェクトを右クリックし、[追加 > 新しいファイル]をクリックします。

4.13　[新しいファイル]画面から、[General > 空のクラス]を選択し、名前を「PhoneTranslator」と付け、[新規]ボタンをクリックします。



4.14　[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。

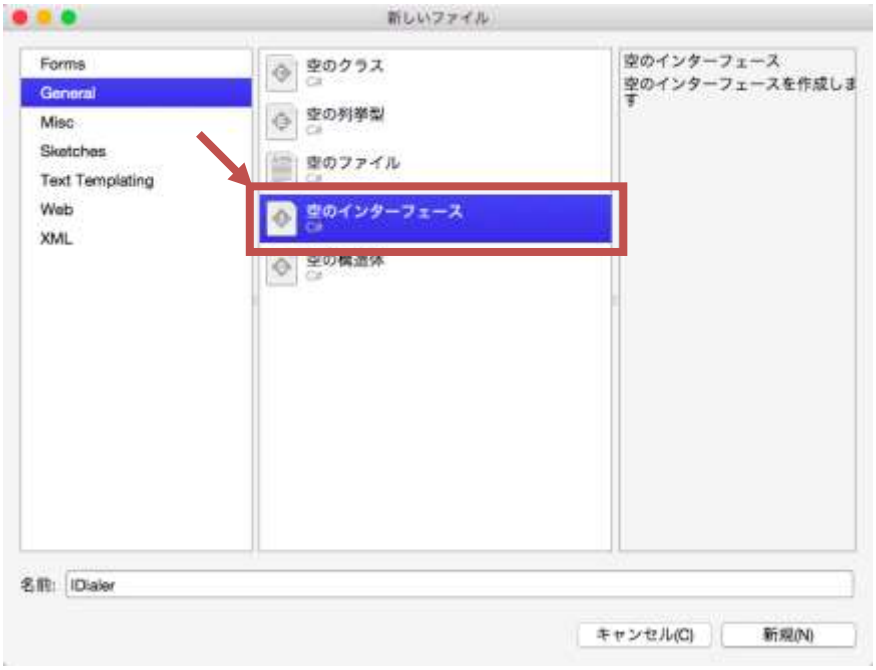
```
using System.Text;

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;
        }
    }
}
```

[⌘＋S]を押して変更を保存します。

4.15　[ソリューション]パッドで、[Phoneword]ソリューションを右クリックし、[追加 > 新しいファイル]をクリックします。

4.16　[新しいファイル]画面から、[General > 空のインターフェイス] を選択し、新しいファイルの名前を「IDialer」と付け、[新規]ボタンをクリックします。



4.17　[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。

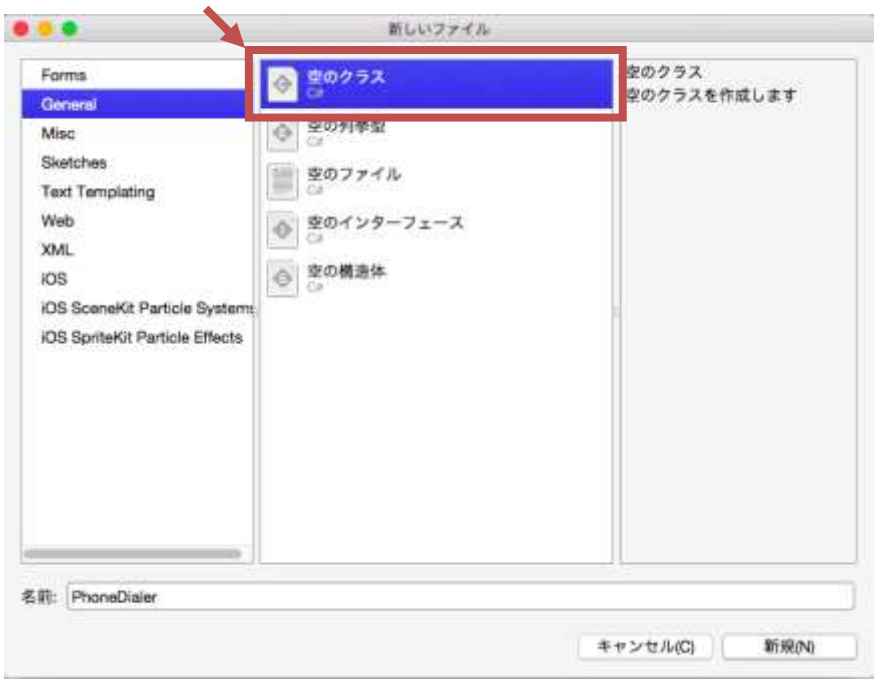
```
using System;

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

[⌘＋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、iOS の実装のみを行います。

4.18　[ソリューション]パッドで、iOS プロジェクト[Phoneword.iOS]プロジェクトを右クリックし、[追加 ＞ 新しいファイル]をクリックします。

4.19　[新しいファイル]画面から、[General ＞ 空のクラス]を選択し、名前を「PhoneDialer」と付け、[新規]ボタンをクリックします。



4.20　[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 iOS 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 Foundation;
using Phoneword.iOS;
using UIKit;
using Xamarin.Forms;

[assembly: Dependency(typeof(PhoneDialer))]

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

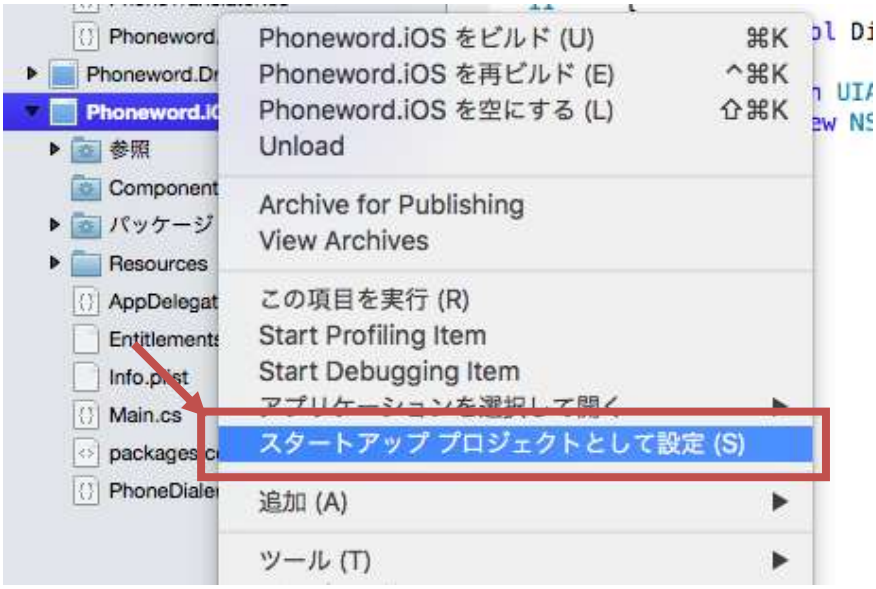
```
}
```

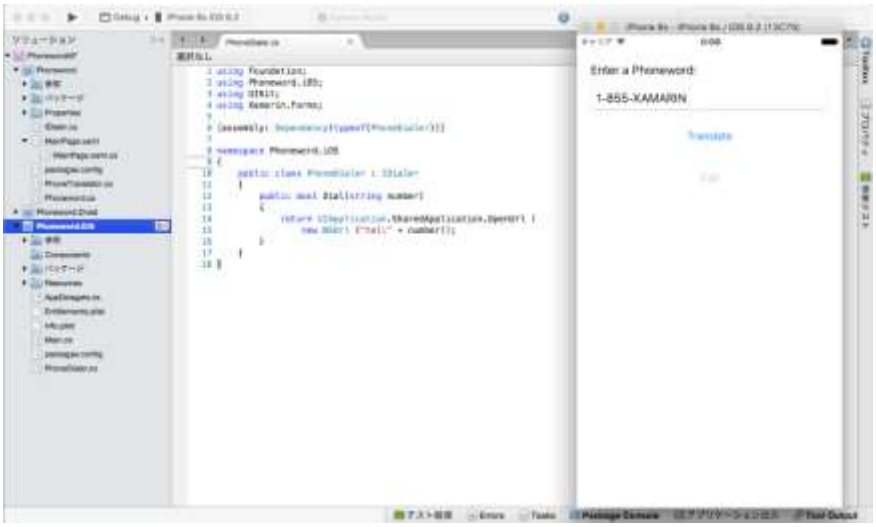
[⌘＋S]を押して変更を保存します。

4.21　[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。

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。

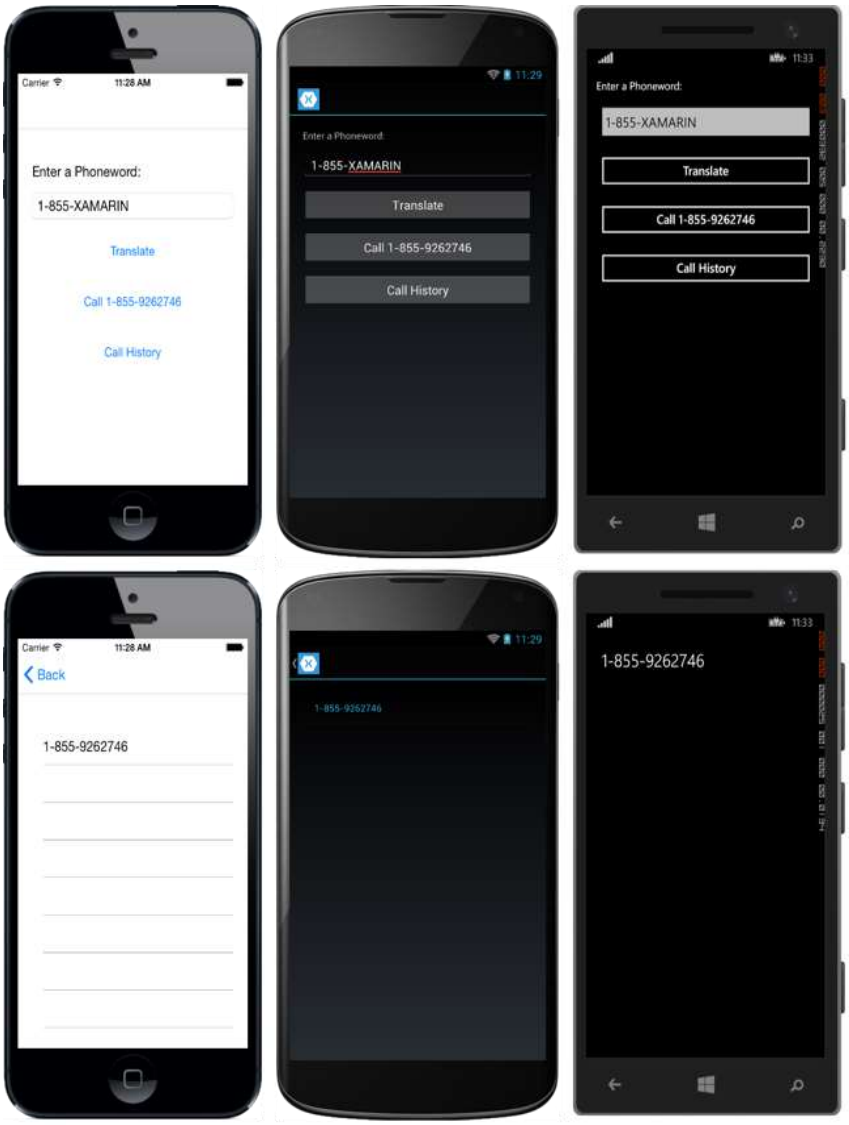
4.22　Xamarin Studio で[Phoneword.iOS]を右クリックして、[スタートアッププロジェクトとして設定]をクリックし、[▶]ボタンをクリックし、アプリケーションを起動します。




Enter a Phoneword:
1-855-XAMARIN
Translate

5　Xamarin.Forms Multiscreen Quickstart

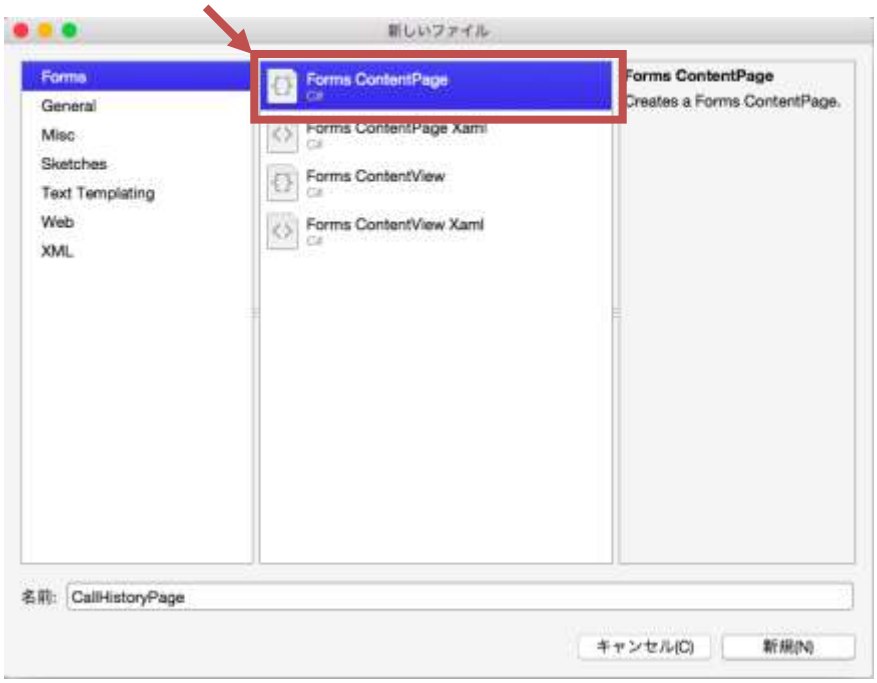
このセクションでは、先ほど作成した Xamarin.Forms アプリケーションをマルチスクリーンに対応させます。完成図は次のようになります。



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。

5.2　[ソリューション]パッドで[Phoneword]プロジェクトを右クリックし、[追加 > 新しいファイル]をクリックします。

5.3　[新しいファイル] 画面から、[Forms > Forms ContentPage]を選択し、名前を「CallHistoryPage」と付け、[新規]ボタンをクリックします。



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,
                    }
                }
            };
        }
    }
```

```
}
```

[⌘＋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]パッドで[Phoneword.cs]をダブルクリックして開きます。

5.6 [Phoneword.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 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());
        }
        ...
        ...
        ...
```

```
    }
}
```

[⌘＋S]を押して変更を保存します。

5.7　[ソリューション]パッドで[MainPage.xaml]をダブルクリックして開きます。

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。太字が追加部分です。

```
<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>
```

[⌘＋S]を押して変更を保存します。

5.9　[ソリューション]パッドで[MainPage.xaml.cs]をダブルクリックして開きます。

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。（If 文の後の{}を忘れないようにしてください。）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());
        }
    }
}
```

[⌘＋S]を押して変更を保存します。

5.11 Xamarin Studio のメニューから[ビルド ＞ 全てビルド] 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 Xamarin Studio で Phoneword.iOS を[スタートアッププロジェクトとして設定]でスタートアッププロジェクトとし、[▶]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 iOS アプリの作成方法、Xamarin.Forms で iOS アプリの作成方法を学びました。UWP アプリのビルドには Windows が必要ですので、本ガイドでは扱っておりません。Android も試してみたい方は http://www.xlsoft.com/jp/products/xamarin/support.html の「初めての Xamarin.Android アプリケーション開発 - 入門ガイド」(http://www.xlsoft.com/jp/products/xamarin/android_hello_world.html) などを参考に是非トライしてみてください。