Xamarin.Forms Quick Start Hands On Lab

C#ではじめるモバイルアプリ開発ハンズオン -Xamarin 編-

2016 年 12 月

Microsoft

内容

概要

ここでは、Visual Studio を使用した Xamarin.Forms アプリケーション開発の基本的な部分を紹介します。Xamarin.Forms 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。

目的

このラボでは、以下の方法を示します。

- Xamarin.Forms を用いたクライアントアプリを実装する
- プラットフォーム毎の実装を行う (Dependency Service)
- ページ遷移を実装する
- MVVM パターンを用いた、RSS リーダーアプリを実装する

システム要件

このラボを完了するには、以下の環境やツールが必要です。

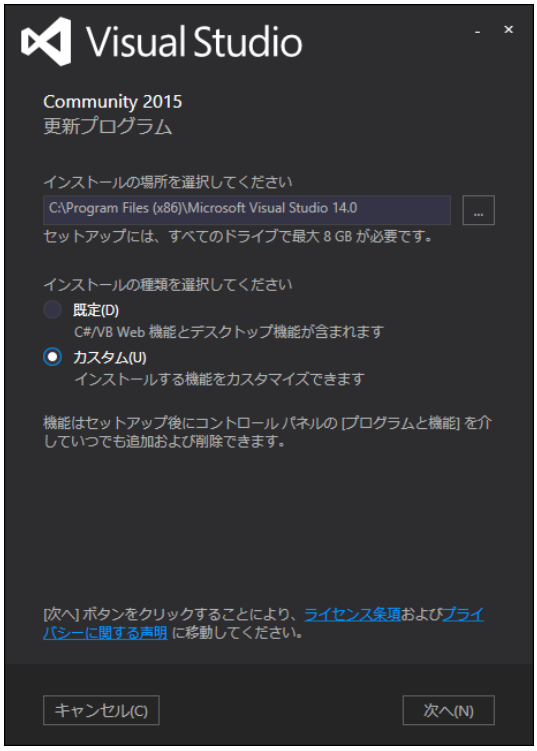
- Microsoft Windows 10 x64 Professional または Enterprise エディション
  (Hyper-V による、Windows 10 Mobile および Android エミュレータを使用するため)
- Microsoft Visual Studio 2015 Update 3 および クロスプラットフォーム開発環境
- Android デバイス（オプショナル）
- Windows 10 Mobile デバイス（オプショナル）
- Mac OS が動作する PC（オプショナル）
- iOS デバイス（オプショナル）

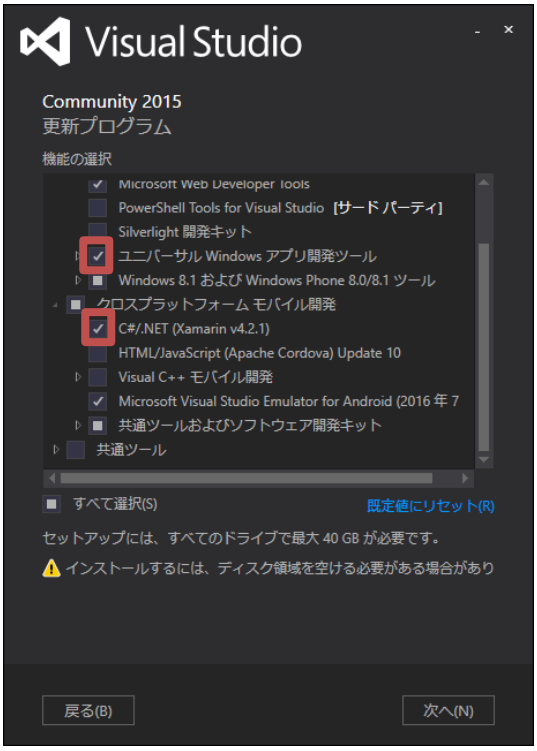
セットアップ

このラボ用にコンピューターの準備を整えるには、以下の手順を実行する必要があります。

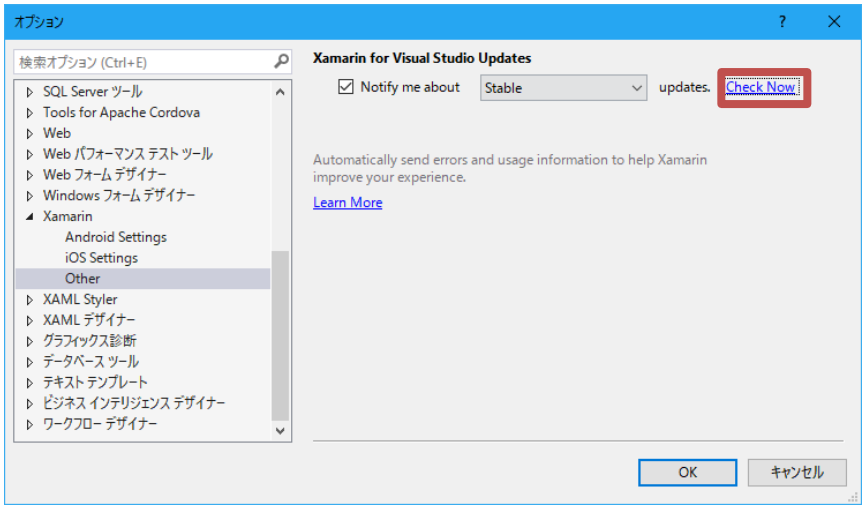
1. Microsoft Windows 10 x64 Professional または Enterprise エディションをインストールします。

2. Microsoft Visual Studio 2015 Update 3 をインストールします。[カスタム] インストールを選択し、[機能の選択] の一覧で、下記を選択し、[次へ] ボタンをクリックします。
   - [Windows 開発と Web 開発] → [ユニバーサル Windows アプリ開発ツール]
   - [クロスプラットフォーム モバイル開発] → [C#/.NET (Xamarin vX.X.X)]





3. Visual Studio 2015 Update 3 を起動します。[ツール > オプション] で [オプション] ダイアログを開き、[Xamarin > Other] の [Check Now] リンクをクリックして Xamarin を最新版にアップデートしてください。



演習

このハンズオン ラボは、以下の演習から構成されています。

1. — Xamarin.Forms Quick start
2. — Xamarin.Forms MultiScreen Quick start
3. — MVVM パターンを用いた、RSS リーダーアプリの作成

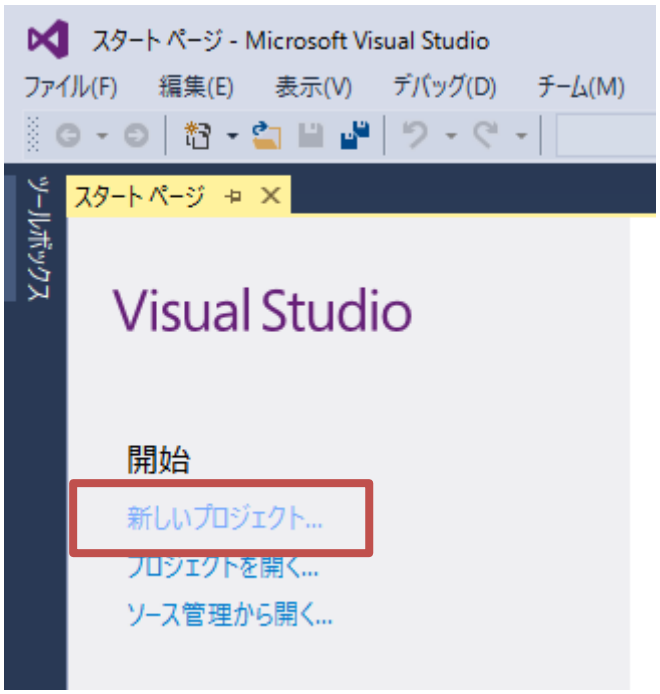
ラボの推定所要時間: **60 ～ 90 分**

演習 1: Xamarin.Forms Quick start

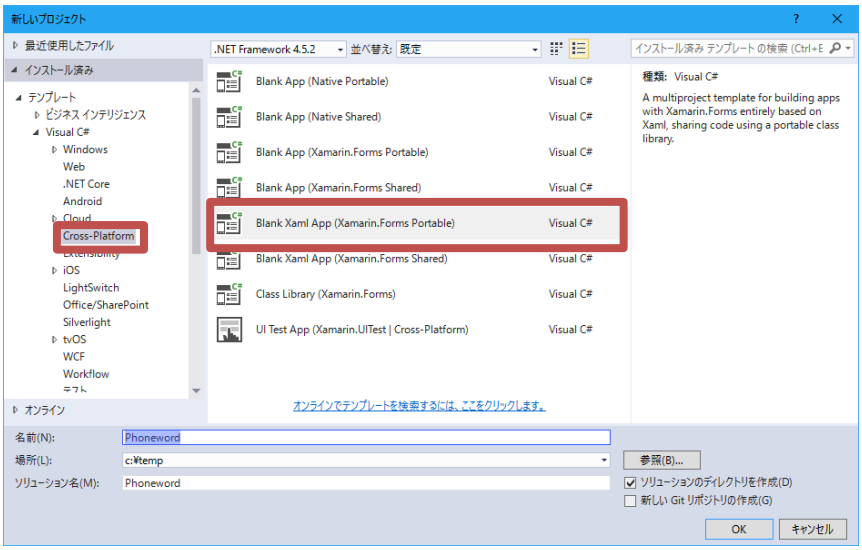
この演習では、ユーザーが入力した英数字の電話番号を数字の電話番号に変換し、その番号に電話するアプリケーションを作成します。

タスク 1 – プロジェクトの新規作成

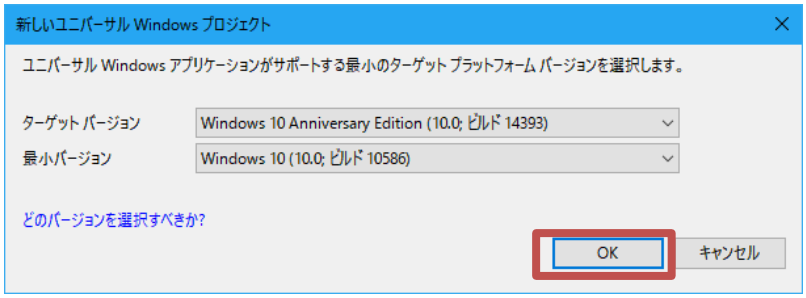
1. Visual Studio を起動し、[スタートページ > 新しいプロジェクト] をクリックして、新しいソリューションを作成します。



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



3. プロジェクト作成中に、UWP プロジェクトのバージョンを指定するウィンドウが表示されますが、そのまま [OK] ボタンをクリックします。

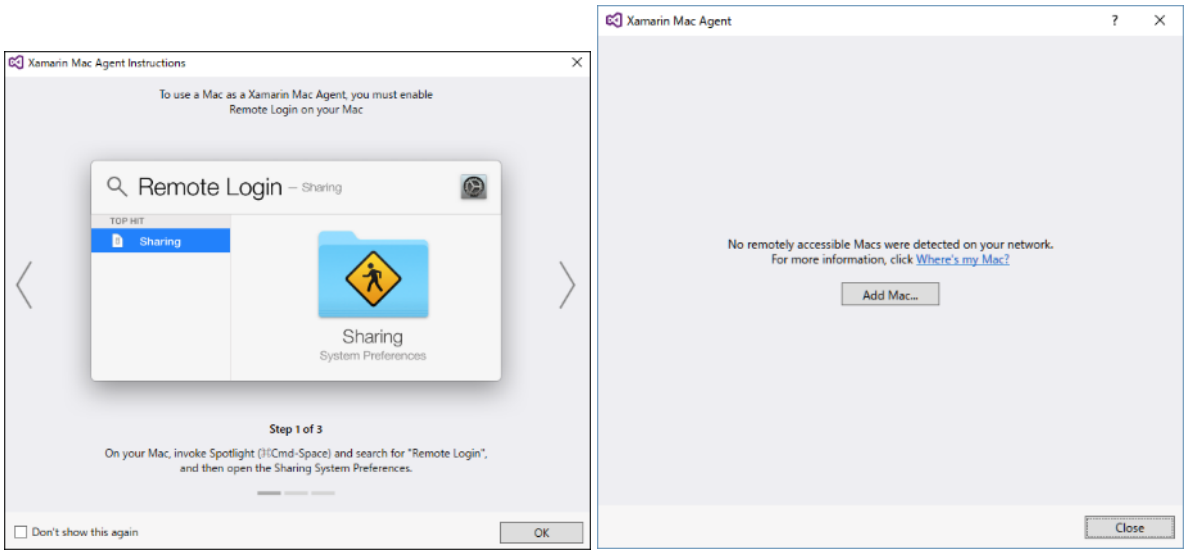


このソリューションは 6 つのプロジェクトで構成されています。

- ✓ Phoneword (移植可能) – 共有コードを配置するための Portable Class Library (PCL)
- ✓ Phoneword.Droid - Xamarin.Android アプリケーション
- ✓ Phoneword.iOS - Xamarin.iOS アプリケーション
- ✓ Phoneword.UWP (Universal Windows) - Windows 10 UWP アプリケーション
- ✓ Phoneword.Windows (Windows 8.1) – Windows 8.x に対応した Windows ストアアプリ
- ✓ Phoneword.Windows (Windows Phone 8.1) – Windows Phone 8.x に対応した Windows Phone ストアアプリ

今回の演習では、Android、iOS、UWP を対象とし、Windows 8.1 および Windows Phone 8.1 は対象外とします。

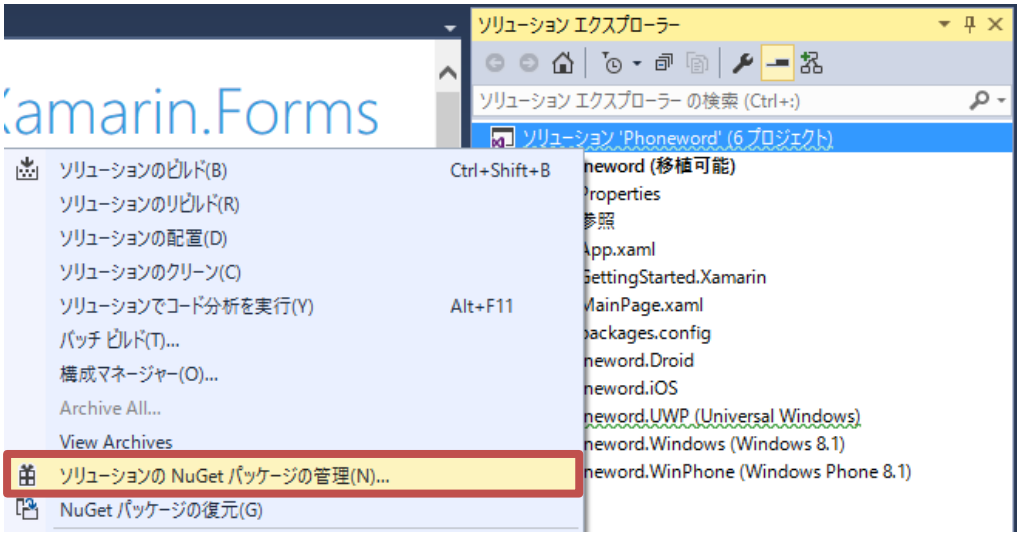
Xamarin.iOS プロジェクトのビルドには Mac の環境が必要となりますが、無い場合は、Xamarin Mac Agent のウィンドウが表示されます。Close または画面右上の × を押して画面を閉じ、次に進みます。環境がある場合は、[付録] のセットアップ手順に従い、セットアップを行ってください。



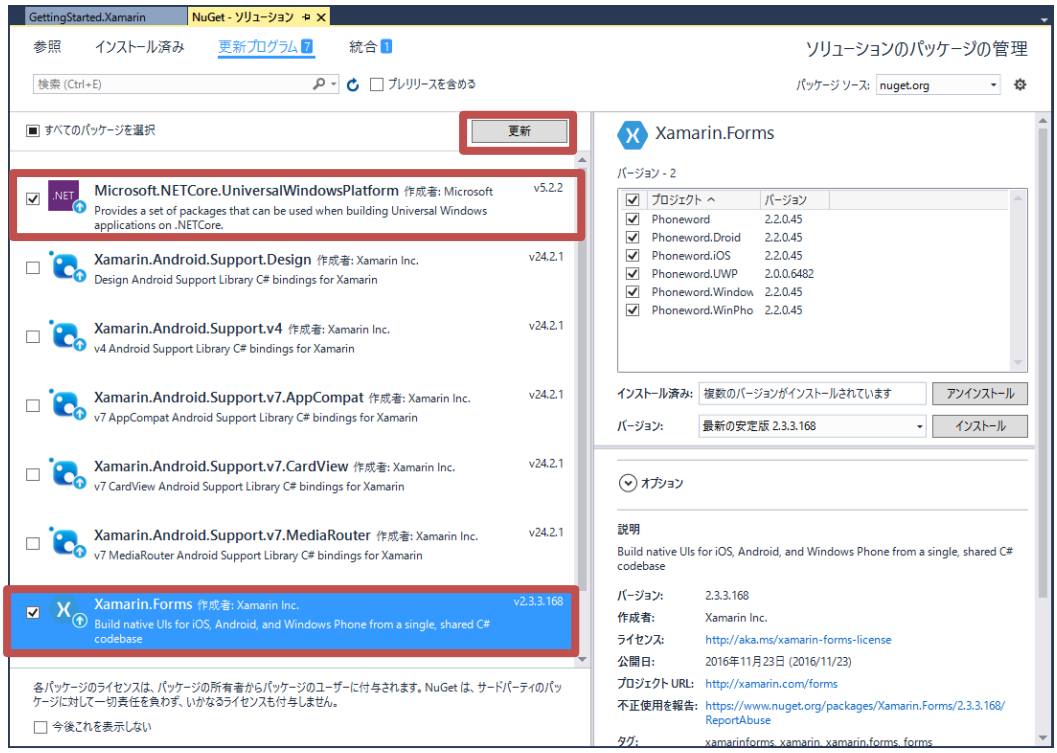
タスク 2 – NuGet パッケージの更新

全てのプロジェクトで必要な NuGet パッケージはソリューション テンプレートにてインストール済みですが、テンプレート作成時点の古いバージョンになっています。そこで、利用している NuGet パッケージを最新バージョンに更新を行います。

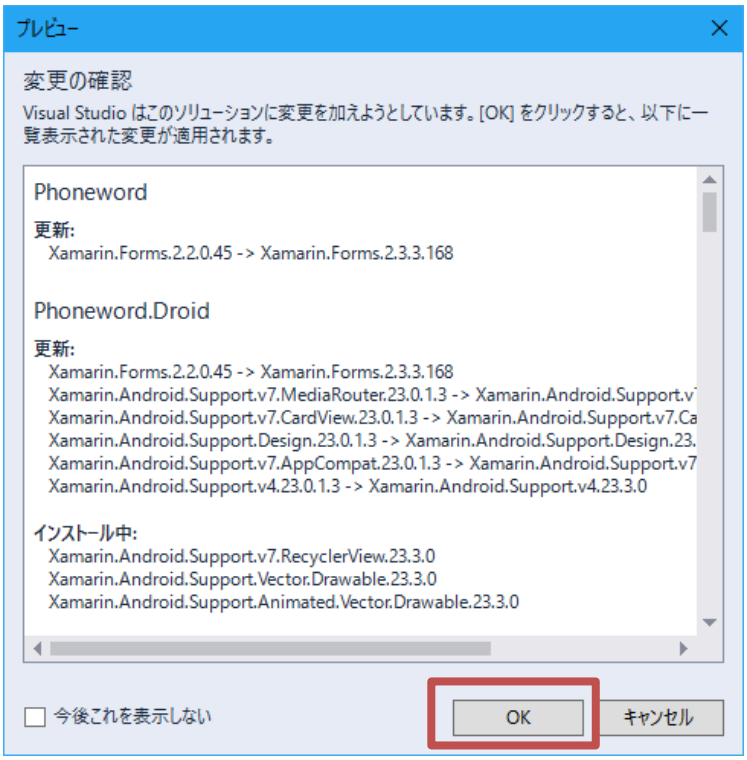
1. パッケージの更新は、[ソリューション エクスプローラー] より [Phoneword] ソリューションを右クリックし、[ソリューションの NuGet パッケージの管理] をクリックします。



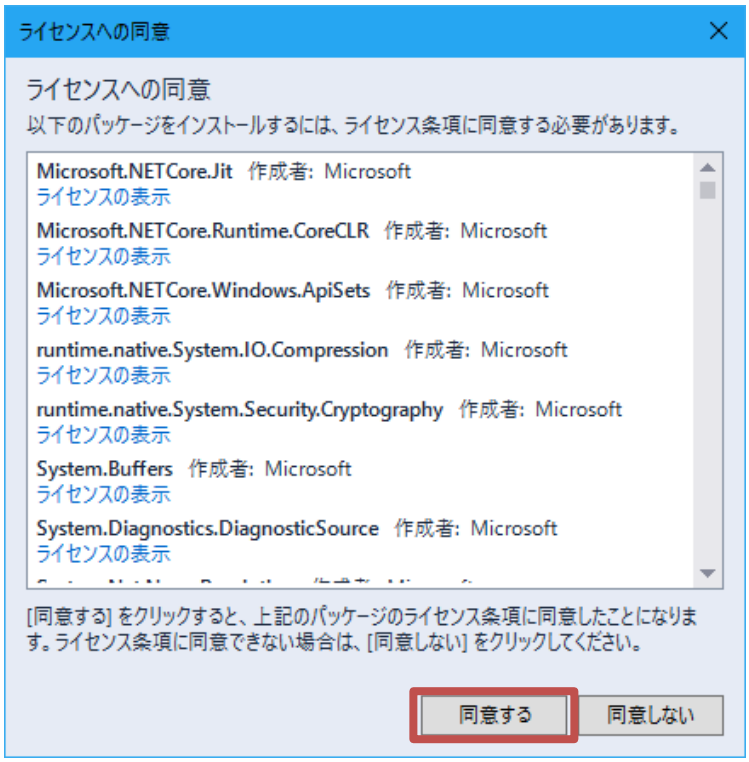
2. [NuGet ソリューションのパッケージの管理] ダイアログで [更新プログラム] タブをクリックし、**[Microsoft.NETCore.UniversalWindowsPlatform] と [Xamarin.Forms] にチェックをして**、[更新] ボタンをクリックします。



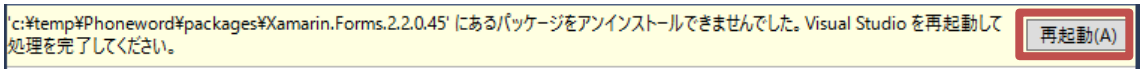
3. [更新の確認] ダイアログが表示されますので、[OK] ボタンをクリックします。



4. [ライセンスへの同意] ダイアログが表示されるので、内容を確認し、[同意する] ボタンをクリックします。



5. [NuGet ソリューションのパッケージの管理] ダイアログの上部に、下記のような表示が出た場合は、[再起動] ボタンをクリックして、Visual Studio を再起動します。



タスク 3 – XAML の実装

英数字の電話番号の入力や、電話番号への変換ボタン、発話ボタンなどの UI を XAML で実装します。

1. [ソリューション エクスプローラー] より、Phoneword プロジェクトにある MainPage.xaml ファイルを開きます。開いたら、すべてのコードを削除し、以下のコードで置き換えます。

更新前

MainPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:Phoneword"
             x:Class="Phoneword.MainPage">

  <Label Text="Welcome to Xamarin Forms!"
         VerticalOptions="Center"
```

```
            HorizontalOptions="Center" />

</ContentPage>
```

更新後

MainPage.xaml

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

Ctrl+S キーを押して、保存をしてください。

タスク 4 – コードビハインドの実装

続いて、ボタンがタップされた時の動作を実装します。

1. [ソリューション エクスプローラー] より、Phoneword プロジェクトにある MainPage.xaml を展開し、MainPage.xaml.cs を開きます。開いたら、すべてのコードを削除し、以下のコードで置き換えます。

   更新前

   MainPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using Xamarin.Forms;

namespace Phoneword
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

更新後

MainPage.xaml.cs

```
using System;
using Xamarin.Forms;

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

        public MainPage()
        {
            InitializeComponent();
        }

        void OnTranslate(object sender, EventArgs e)
        {
            translatedNumber =
Core.PhonewordTranslator.ToNumber(phoneNumberText.Text);
            if (!string.IsNullOrWhiteSpace(translatedNumber))
            {
                callButton.IsEnabled = true;
                callButton.Text = $"Call {translatedNumber}";            }
            else
            {
                callButton.IsEnabled = false;
                callButton.Text = "Call";
            }
        }

        async void OnCall(object sender, EventArgs e)
        {
            if (await this.DisplayAlert(
                    "Dial a Number",
                    $"Would you like to call {translatedNumber} ?",
                    "Yes",
```

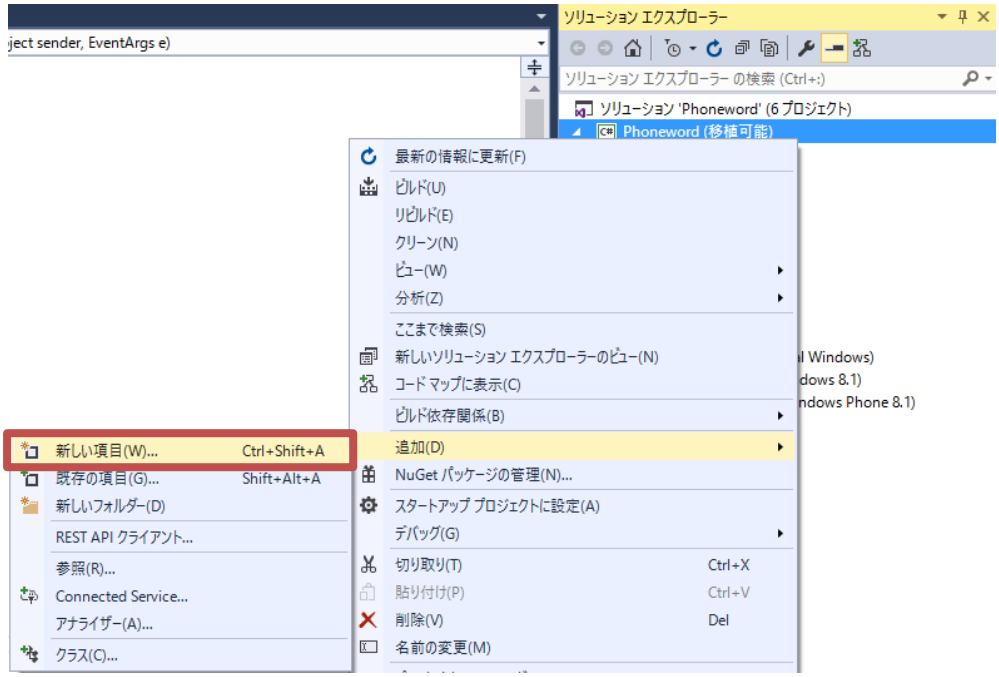
```
                    "No"))
            {
                var dialer = DependencyService.Get<IDialer>();
                if (dialer != null)
                    dialer.Dial(translatedNumber);
            }
        }
    }
}
```

Ctrl+S キーを押して、保存をしてください。

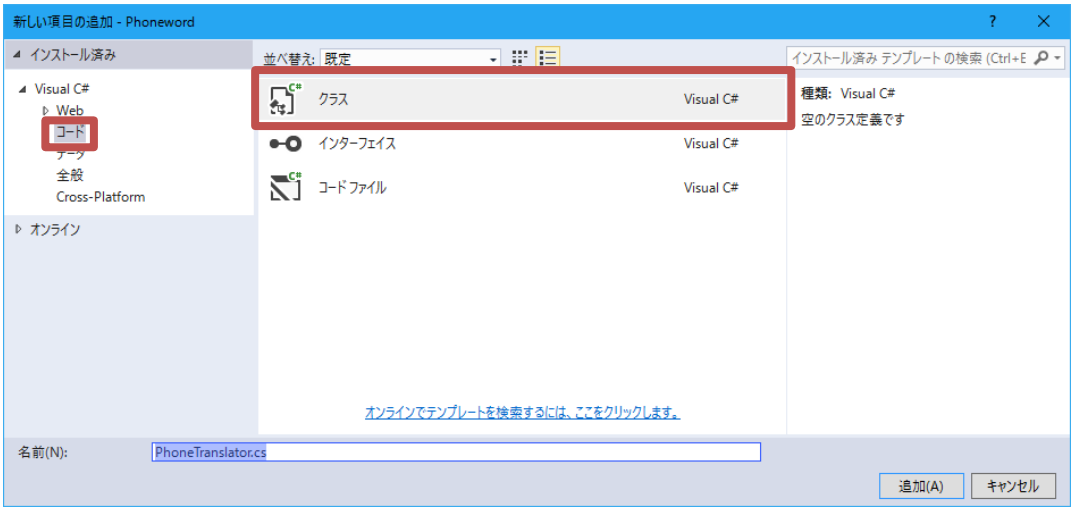
タスク 5 – 電話番号変換ロジックの実装

続いて、入力された文字列を電話番号に変換するロジックを実装します。

1. [ソリューション エクスプローラー] より、Phoneword プロジェクトを右クリックし、[追加 > 新しい項目] を選択します。



2. [新しい項目の追加] ウィンドウで、左ペインから [Visual C# > コード] を選択します。[クラス] を選択し、名前を [PhoneTranslator.cs] と入力して、[追加] ボタンをクリックします。



3. 開かれた [PhoneTranslator.cs] の内容をすべて削除し、以下のコードで置き換えます。

更新前

PhoneTranslator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Phoneword
{
    class PhoneTranslator
    {
    }
}
```

更新後

PhoneTranslator.cs

```
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 キーを押して、保存をしてください。

タスク 6 – XAML コンパイル設定の定義

続いて、XAML コンパイル設定の定義を追加します。既定では、実行時に XAML を読み込み、解釈が行われるため、パフォーマンスが悪くなる可能性があります。そこで、ビルド時に XAML も併せてコンパイル済みにすることで、アプリケーションを高速に起動させることが可能となります。また同時に、XAML の実装ミスをコンパイル時に発見することにもつながります。

1. [ソリューション エクスプローラー] より、Phoneword プロジェクトにある App.xaml を展開し、App.xaml.cs を開きます。下記のように書き換えます。追加された行を太字にしています。

更新前

App.xaml.cs

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

using Xamarin.Forms;

namespace Phoneword
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            MainPage = new Phoneword.MainPage();
        }
        // ・・・・省略・・・・
    }
}
```

更新後

App.xaml.cs

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

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

[assembly:XamlCompilation(XamlCompilationOptions.Compile)]

namespace Phoneword
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            MainPage = new Phoneword.MainPage();
        }
        // ・・・・省略・・・・
    }
}
```

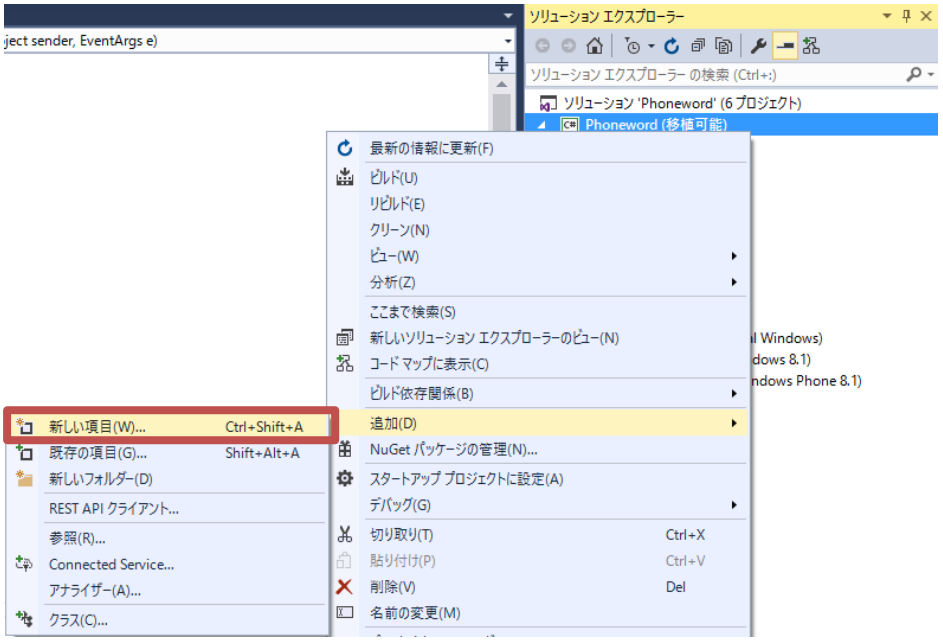
Ctrl+S キーを押して、保存をしてください。

ここまでの実装で、UI の表示、入力された文字列の電話番号への変換までが完成しました。

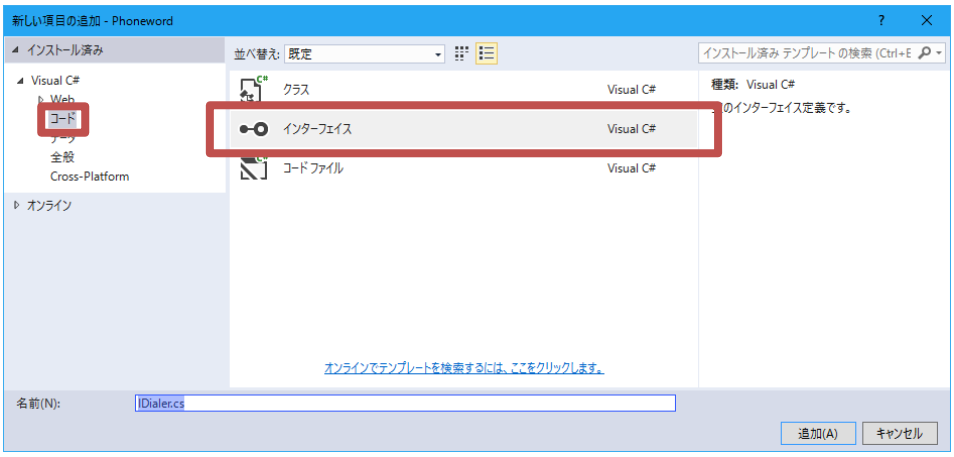
タスク 7 – IDialer の実装

続いて、発話をするための実装を行っていきます。まずは共通コードとしてインターフェイスを定義します。

1. [ソリューション エクスプローラー] より、Phoneword プロジェクトを右クリックし、[追加 > 新しい項目] を選択します。



2. [新しい項目の追加] ウィンドウで、左ペインから [Visual C# > コード] を選択します。[インターフェイス] を選択し、名前を [IDialer.cs] と入力して、[追加] ボタンをクリックします。



3. 開かれた [IDialer.cs] の内容を、以下のコードで置き換えます。

更新前

IDialer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Phoneword
{
    interface IDialer
    {
    }
}
```

更新後

IDialer.cs

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

Ctrl+S キーを押して、保存をしてください。

ここまでで共通の実装は完成しました。この後は、プラットフォーム毎に、発話するためのコードを Dependency Service という機能を利用して実装していきます。

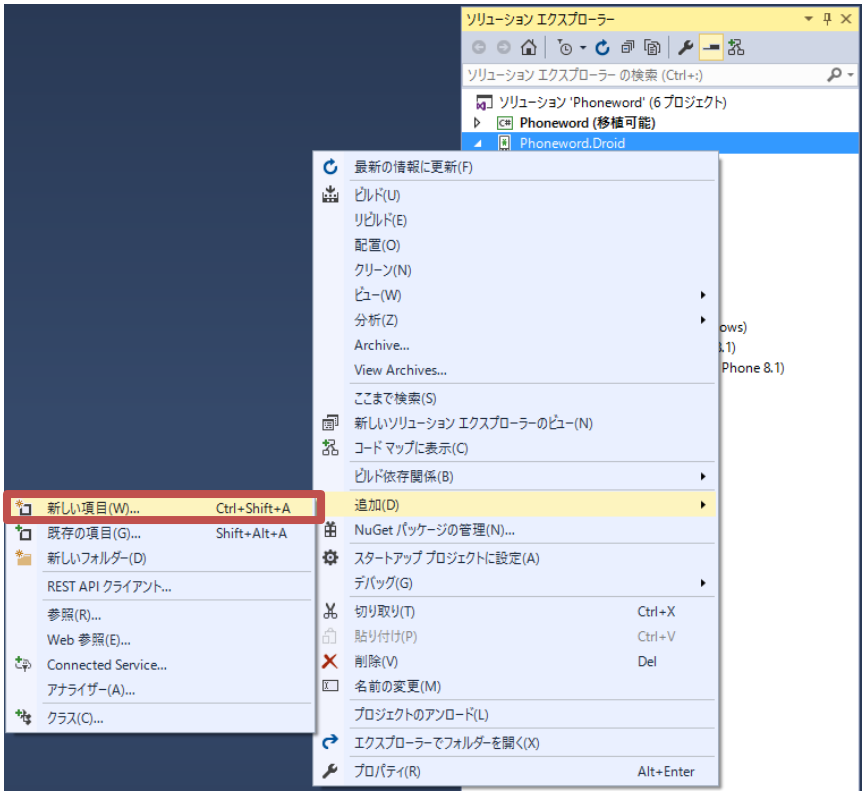
タスク 8 – IDialer のプラットフォーム毎の実装

続いて、Xamarin.Forms の Dependency Service の機能を利用して、プラットフォーム毎に発話するためのコードを実装していきます。

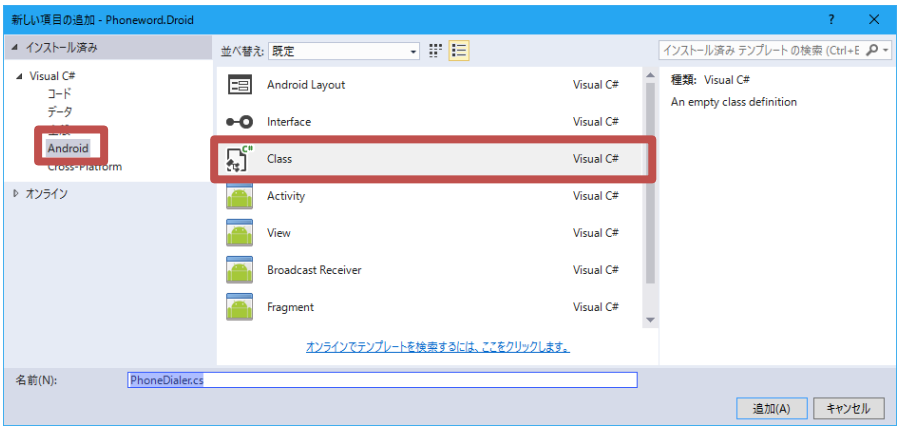
Android

まずは、Android の発話コードを実装します。Android では、Intent という機能を利用して、電話アプリを起動することができます。

1. [ソリューション エクスプローラー] より、Phoneword.Droid プロジェクトを右クリックし、[追加 > 新しい項目] を選択します。



2. [新しい項目の追加] ウィンドウで、左ペインから [Visual C# > Android] を選択します。[Class] を選択し、名前を [PhoneDialer.cs] と入力して、[追加] ボタンをクリックします。



3. 開かれた [PhoneDialer.cs] の内容を、以下のコードで置き換えます。

更新前

PhoneDialer.cs

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

using Android.App;
```

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

namespace Phoneword.Droid
{
    class PhoneDialer
    {
    }
}
```

更新後

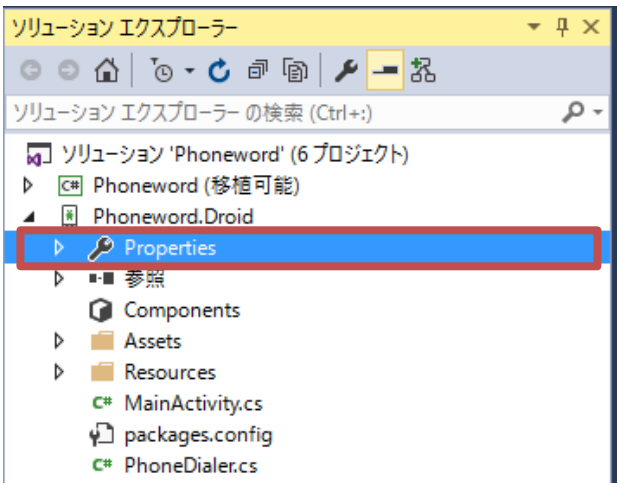
PhoneDialer.cs

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

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

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

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

            return false;
        }

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

            var list = packageManager.QueryIntentServices(intent, 0)
```

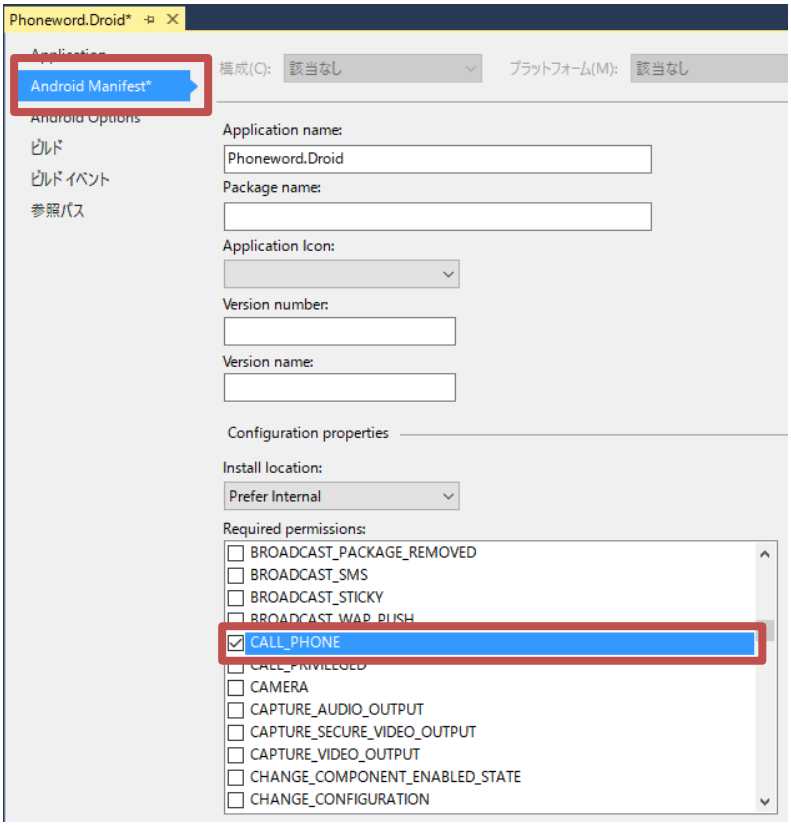
```
                .Union(packageManager.QueryIntentActivities(intent, 0));

            if (list.Any())
                return true;

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

Ctrl+S キーを押して、保存をしてください。

4. Android で発話をするために、アプリケーションの権限を有効にする必要があります。[ソリューション エクスプローラー] から Phoneword.Droid プロジェクトを展開し、[Properies] をダブルクリックし、プロジェクトプロパティ ウィンドウを表示します。



5. 左ペインから [Android Manifest] をクリックし、[Required permissions] から [CALL_PHONE] を有効にします。

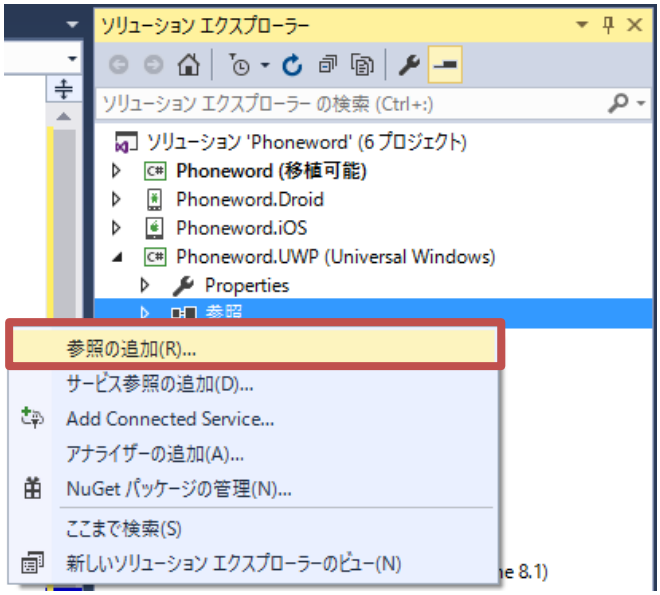


Ctrl+S キーを押して、保存をしてください。

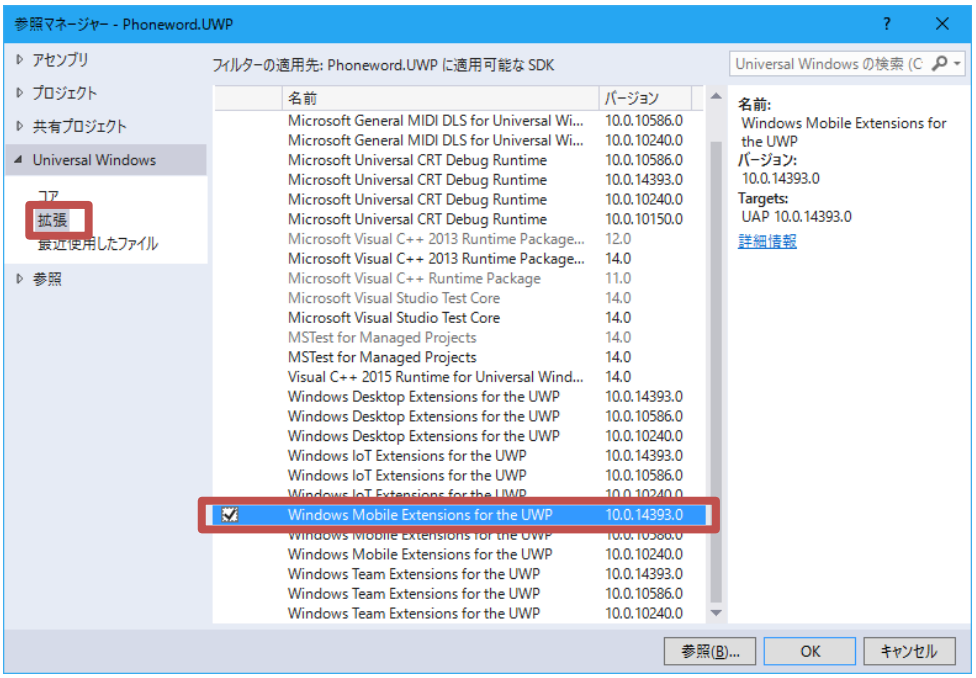
Windows (UWP)

続いて、UWP アプリの実装を行っていきます。UWP アプリは、Windows 10 ファミリーの Desktop 、Mobile など様々なプラットフォームで動作することができますが、発話機能である PhoneCallManager クラスは Mobile のみの機能となります。

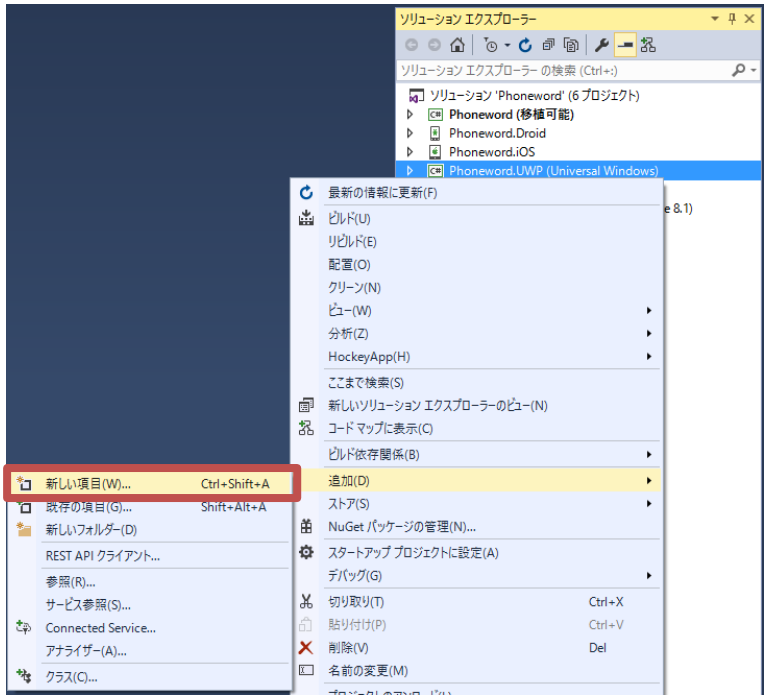
1. PhoneCallManager クラスを呼び出すため、Mobile 用の拡張機能を参照に追加します。[ソリューション エクスプローラー] から Phoneword.UWP プロジェクトを展開、[参照] を右クリックし、[参照の追加] をクリックします。



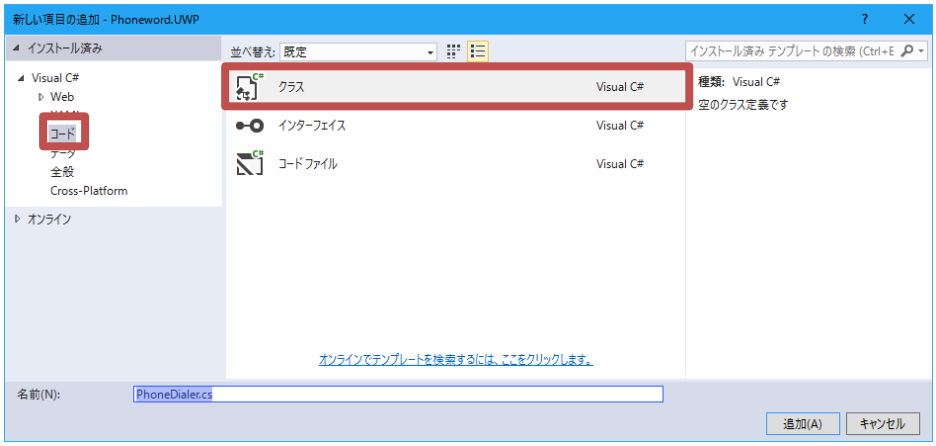
2. [参照マネージャー] ウィンドウが起動したら、左ペインのツリーより [Universal Windows > 拡張] をクリックし、[Windows Mobile Extensions for UWP] にチェックを入れ、[OK] ボタンをクリックします。



3. [ソリューション エクスプローラー] より、Phoneword.UWP プロジェクトを右クリックし、[追加 > 新しい項目] を選択します。



4. [新しい項目の追加] ウィンドウで、左ペインから [Visual C# > コード] を選択します。[クラス] を選択し、名前を [PhoneDialer.cs] と入力して、[追加] ボタンをクリックします。



5. 開かれた [PhoneDialer.cs] の内容を、以下のコードで置き換えます。

更新前

PhoneDialer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Phoneword.UWP
{
```

```
    class PhoneDialer
    {
    }
}
```

更新後

PhoneDialer.cs

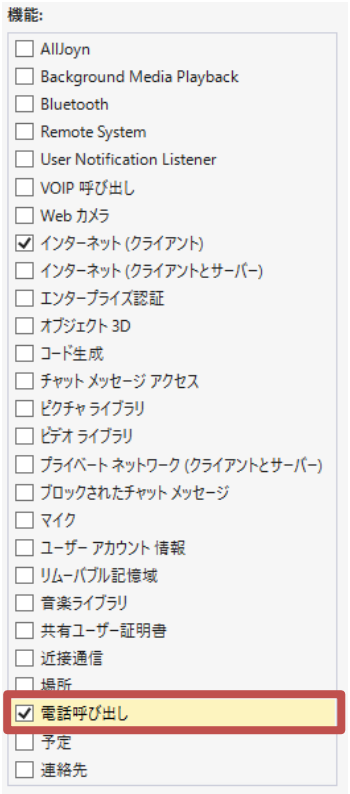
```
using Phoneword.UWP;
using System;
using System.Threading.Tasks;
using Windows.ApplicationModel.Calls;
using Windows.Foundation.Metadata;
using Windows.UI.Popups;
using Xamarin.Forms;

[assembly: Dependency(typeof(PhoneDialer))]
namespace Phoneword.UWP
{
    public class PhoneDialer : IDialer
    {
        bool dialled = false;

        public bool Dial(string number)
        {
            DialNumber(number);
            return dialled;
        }

        async void DialNumber(string number)
        {
            var phoneLine = await GetDefaultPhoneLineAsync();
            if (phoneLine != null)
            {
                phoneLine.Dial(number, number);
                dialled = true;
            }
            else
            {
                var dialog = new MessageDialog("No line found to place the
call");
                await dialog.ShowAsync();
                dialled = false;
            }
        }

        async Task<PhoneLine> GetDefaultPhoneLineAsync()
        {
            // PhoneCallManager クラスが無い場合は呼び出しを行わない
            if (!ApiInformation.IsTypePresent("
Windows.ApplicationModel.Calls.PhoneCallManager")) return null;
```

```
            var phoneCallStore = await PhoneCallManager.RequestStoreAsync();
            var lineId = await phoneCallStore.GetDefaultLineAsync();
            return await PhoneLine.FromIdAsync(lineId);
        }
    }
}
```

Ctrl+S キーを押して、保存をしてください。

6. 続いて、発話機能の権限を追加します。[ソリューション エクスプローラー] から Phoneword.UWP プロジェクトを展開、[Package.appxManifest] をダブルクリックし、アプリケーション マニフェスト デザイナー ウィンドウを表示します。

7. 続いて、[機能] タブを選択し、[電話呼び出し] にチェックを入れます。

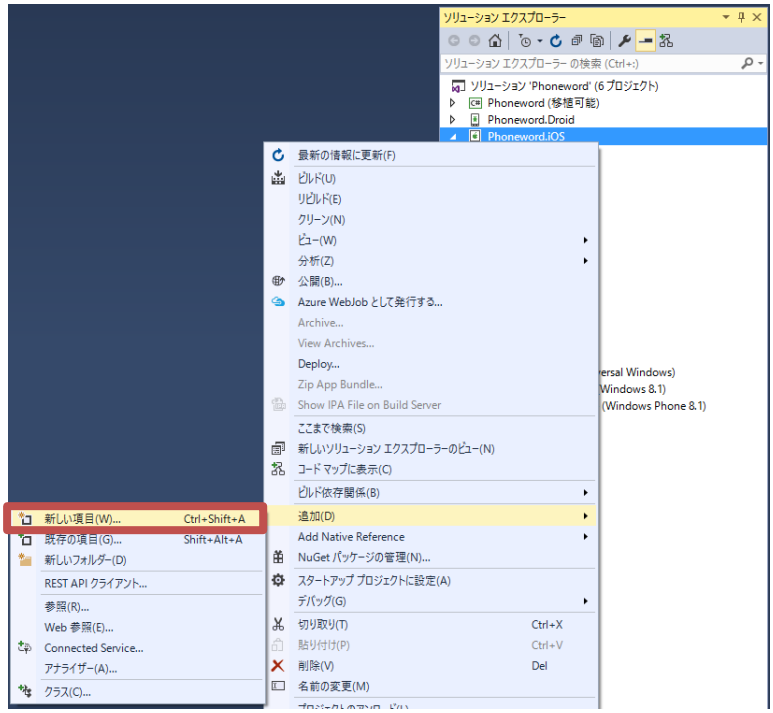


Ctrl+S キーを押して、保存をしてください。

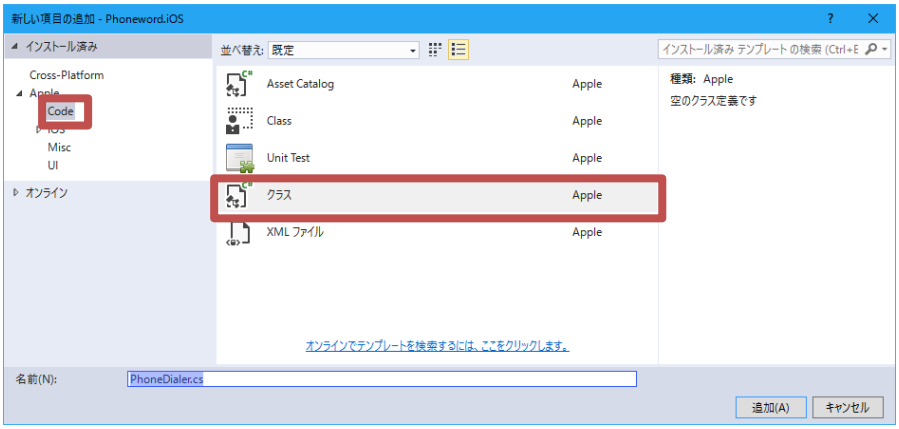
(オプション) iOS

続いて、iOS アプリの実装を行っていきます。iOS アプリは、tel:xxxx-xxxx-xxxx という URL でアプリケーションを起動することで、発話を行うことが可能です。

1. [ソリューション エクスプローラー] より、Phoneword.iOS プロジェクトを右クリックし、[追加 > 新しい項目] を選択します。



2. [新しい項目の追加] ウィンドウで、左ペインから [Apple > Code] を選択します。[クラス] を選択し、名前を [PhoneDialer.cs] と入力して、[追加] ボタンをクリックします。



3. 開かれた [PhoneDialer.cs] の内容を、以下のコードで置き換えます。

更新前

PhoneDialer.cs

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

namespace Phoneword.iOS
{
    class PhoneDialer
    {
```

```
    }
}
```

更新後

PhoneDialer.cs

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

Ctrl+S キーを押して、保存をしてください。

タスク 9 – アプリケーションのデバッグ実行

ここまでのタスクですべての実装が完了しましたので、プラットフォーム毎にアプリケーションをデバッグ実行してみます。

Android

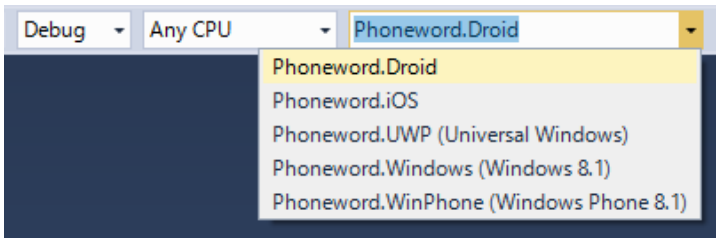
Android アプリの実行はエミュレーターおよび実機の 2 パターンのデバッグが可能です。Android エミュレーターは Google 社が提供している Android Emulator をはじめ、様々なエミュレーターが公開されています。今回の演習では、Microsoft が公開している、Hyper-V で動作する高速な Visual Studio Emulator for Android を利用して実行してみます。

Android 端末でデバッグ実行を行う場合は、事前に開発者向けオプションを表示し、USB デバッグを有効にしておく必要があります。また、Visual Studio を実行している PC と接続する USB ケーブルをご用意ください。

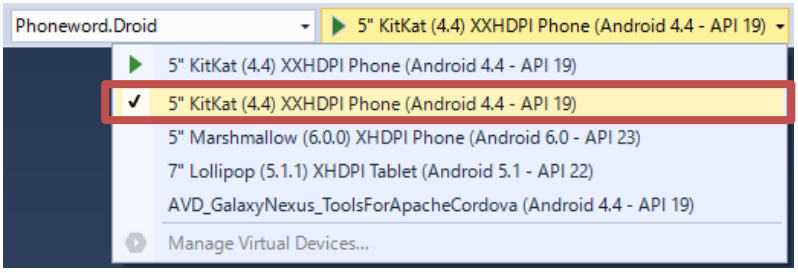
Visual Studio Emulator for Android で実行

1. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。
   - ✓ ソリューション構成：**Debug**
   - ✓ ソリューション プラットフォーム：**Any CPU**

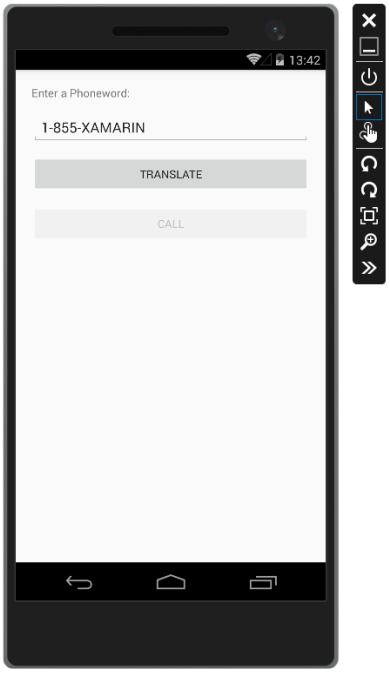
✓ スタートアップ プロジェクト： **Phoneword.Droid**



2. 次に、デバッグ ターゲットから [5" KitKat (4.4) XXHDPI Phone (Android 4.4 – API 19)] を選択し、デバッグを開始します。



3. Android エミュレーターが起動し、アプリケーションが実行できていることを確認します。



(オプション) Android 端末で実行

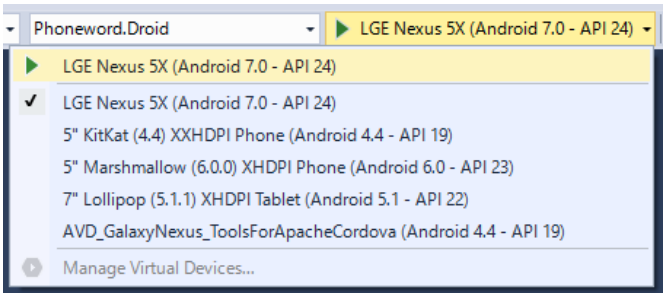
Android 端末をお持ちの方は、下記の手順でデバッグ実行を行うことができます。

1. お手持ちの Android 端末の、[設定 > 開発者向けオプション] を選択し、[USB デバッグ] を有効にします。

メモ

Android 4.2 以降のバージョンでは、[開発者向けオプション] は既定では非表示となっています。表示させるためには、[設定 > 端末情報 > ビルド番号] を 7 回タップすることで、表示されます。

2. Visual Studio を実行している Windows 端末に、USB ケーブルで Android 端末を接続します。
3. デバッグ ターゲットに表示される Android 端末を選択 (下記の例は Nexus 5X) し、デバッグを開始します。

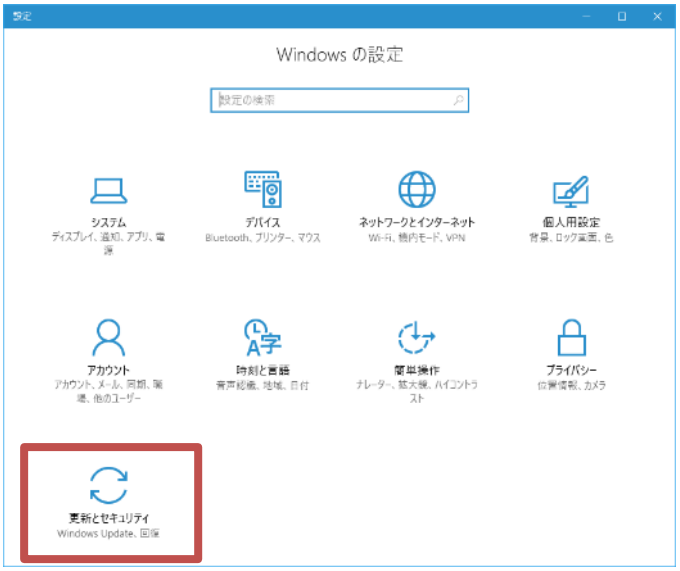


Windows (UWP)

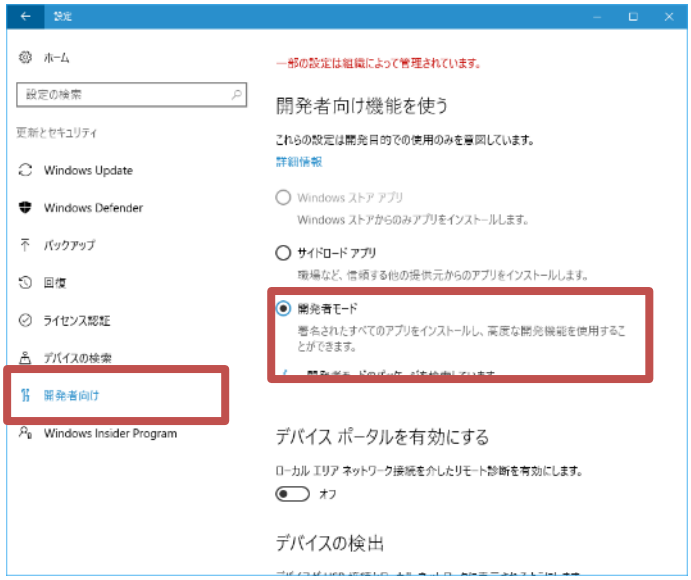
UWP アプリは 、Desktop、Mobile をはじめ、IoT、HoloLens、Surface Hub、Xbox などの、Windows 10 ファミリーの様々なデバイスで動作させることができます。今回の演習では、Desktop および Mobile の 2 つのデバイス ファミリーでデバッグ実行を行います。

ローカル コンピューター で実行

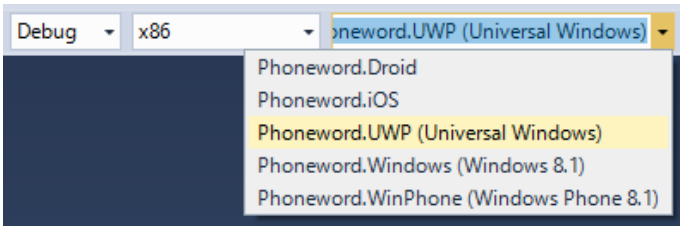
1. UWP をデバック実行するためには、Windows 10 デバイスに対して、開発者モードを有効にする必要があります。ローカル コンピューターで、[設定] アプリを起動し、[更新とセキュリティー] メニューを選択します。



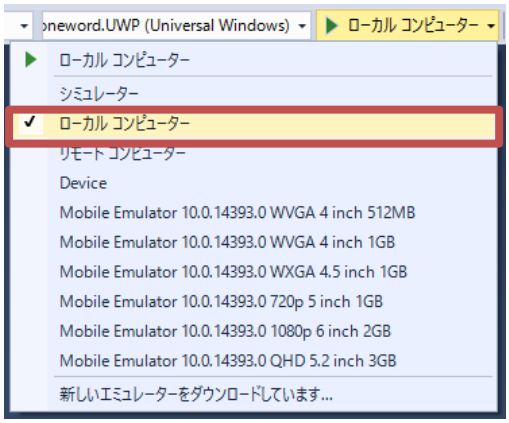
2. [開発者向け] メニューを選択し、[開発者向け機能を使う] の設定を [開発者モード] に変更し、設定アプリを終了します。



3. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。
   - ✓ ソリューション構成：**Debug**
   - ✓ ソリューション プラットフォーム：**x86 または x64**
   - ✓ スタートアップ プロジェクト： **Phoneword.UWP**



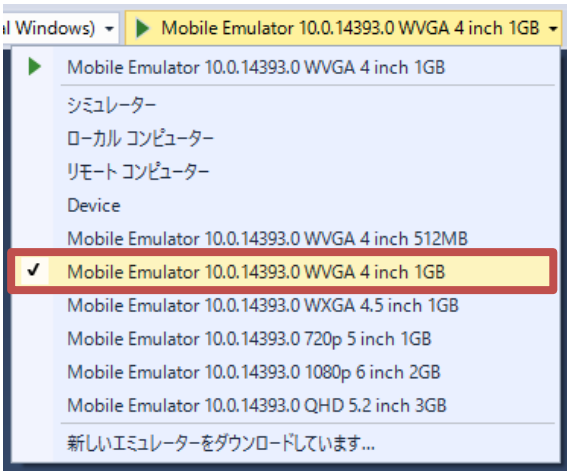
4. 次に、デバッグ ターゲットから [ローカル コンピューター] を選択し、デバッグを開始します。



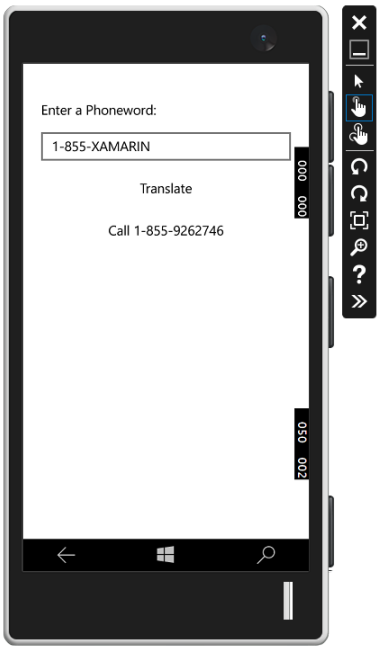
5. Visual Studio を起動しているローカル コンピューター で UWP アプリが起動されることを確認します。

Windows 10 Mobile Emulator で実行

1. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。
   - ✓ ソリューション構成：**Debug**
   - ✓ ソリューション プラットフォーム：**x86**
   - ✓ スタートアップ プロジェクト： **Phoneword.UWP**
2. 次に、デバッグ ターゲットから [Mobile Emulator 10.0.14393.0 WVGA 4 inch 1GB] を選択し、デバッグを開始します。



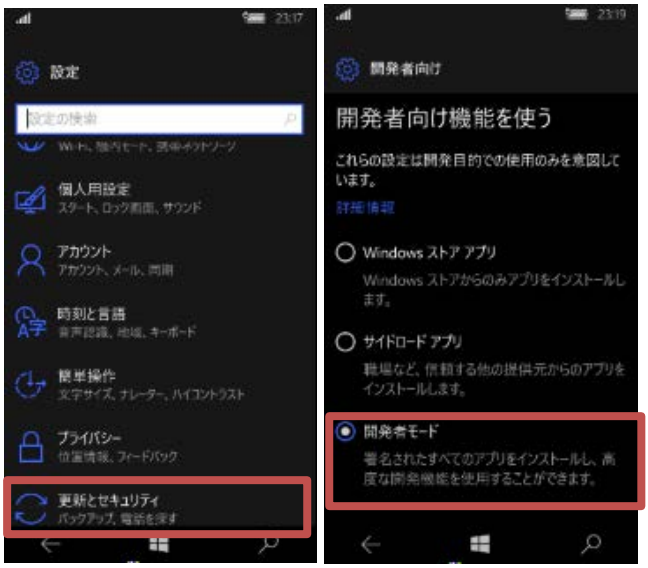
3. Windows 10 Mobile のエミュレーター が起動され、アプリケーションが実行されていることを確認します。



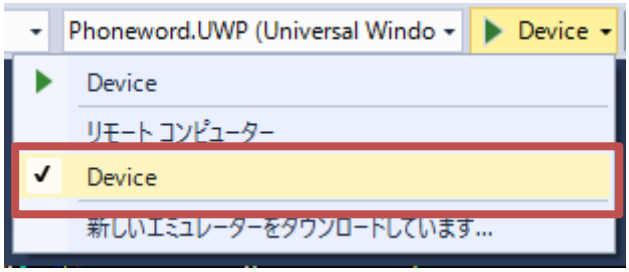
(オプション) Windows 10 Mobile 端末で実行

Windows 10 Mobile 端末のお持ちの方は、実機でのデバッグ実行を行ってみましょう。

1. Windows 10 Mobile 端末で、[設定] アプリを起動し、[更新とセキュリティー] メニューを選択します。
2. [開発者向け] メニューを選択し、[開発者向け機能を使う] の設定を [開発者モード] に変更し、設定アプリを終了します。



3. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。
   - ✓ ソリューション構成：**Debug**
   - ✓ ソリューション プラットフォーム：**ARM**
   - ✓ スタートアップ プロジェクト： **Phoneword.UWP**
4. 次に、デバッグ ターゲットから [Device] を選択し、デバッグを開始します。



5. Windows 10 Mobile 実機でのアプリケーションが実行していることを確認します。

(オプション) iOS

Mac OS が動作する PC と iPhone , iPad などの iOS が動作する実機をお持ちの方は、下記の手順で iOS アプリのデバッグ実行を行うことができます。Mac OS のセットアップと Visual Studio との接続については、付録をご参照ください。

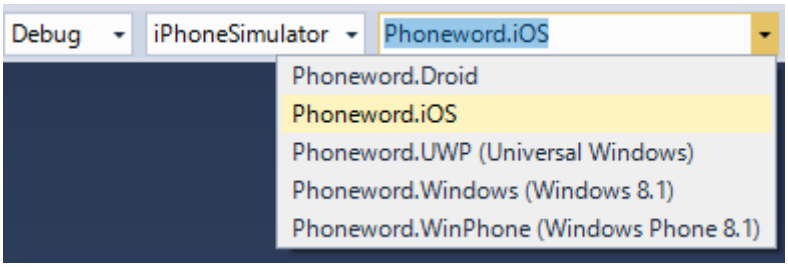
ここでは、Apple 社が提供している、iOS SDK に含まれている iOS Simulator での実行を行います。また、iOS Simulator を Windows 上でリモート操作できるようにするための、iOS Simulator for Windows を利用したデバッグ実行の方法もご紹介いたします。

iOS 端末をお持ちの方は、実機でのデバッグ実行の方法も併せてご紹介いたします。

iOS Simulator で実行

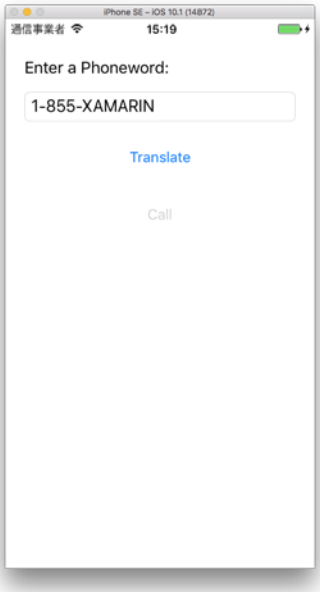
Mac OS 上の iOS Simulator を用いたデバッグ実行は、Visual Studio Comminity エディションを含むすべてのエディションで実行可能です。

1. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。
   - ✓ ソリューション構成：**Debug**
   - ✓ ソリューション プラットフォーム：**iPhoneSimulator**
   - ✓ スタートアップ プロジェクト： **Phoneword.iOS**



2. 次に、デバッグ ターゲットから [iPhone SE iOS 10.1] を選択し、デバッグを開始します。

3. Mac OS 上で iOS Emulator が起動し、iOS アプリケーションを実行できていることを確認します。



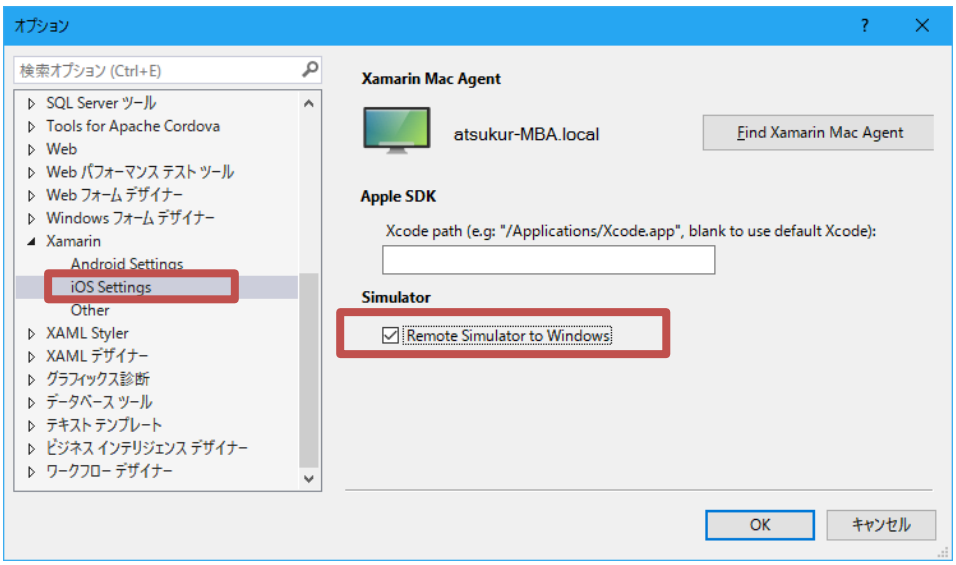
iOS Simulator for Windows で実行

Windows 上に iOS Simulator for Windows をインストールすることで、Mac OS 上で動作する iOS Simulator を Windows 上でリモート操作が可能となります。この機能は Visual Studio Enterprise エディションのみ利用可能です。

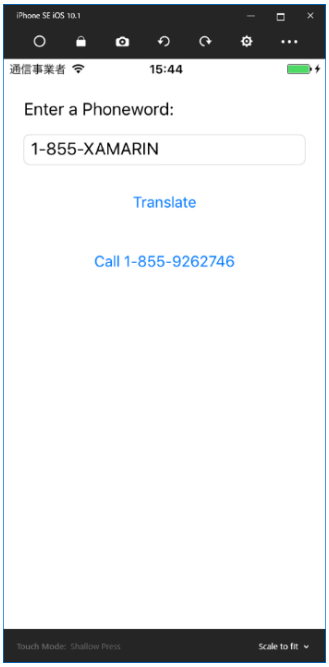
1. 下記 URL より、iOS Simulator for Windows をダウンロードし、インストールを行います。

   https://developer.xamarin.com/guides/cross-platform/windows/ios-simulator/

2. Visual Studio を起動します。[ツール > オプション] で [オプション] ダイアログを開き、[Xamarin > iOS Setting] を表示します。[Remote Simulator for Windows] にチェックを入れ、[OK] ボタンをクリックします。



3. iOS Simurator と同様の設定で、デバッグ実行を行います。Windows 上に iOS Simulator for Windows が起動され、アプリが実行されていることを確認します。



iOS 端末で実行

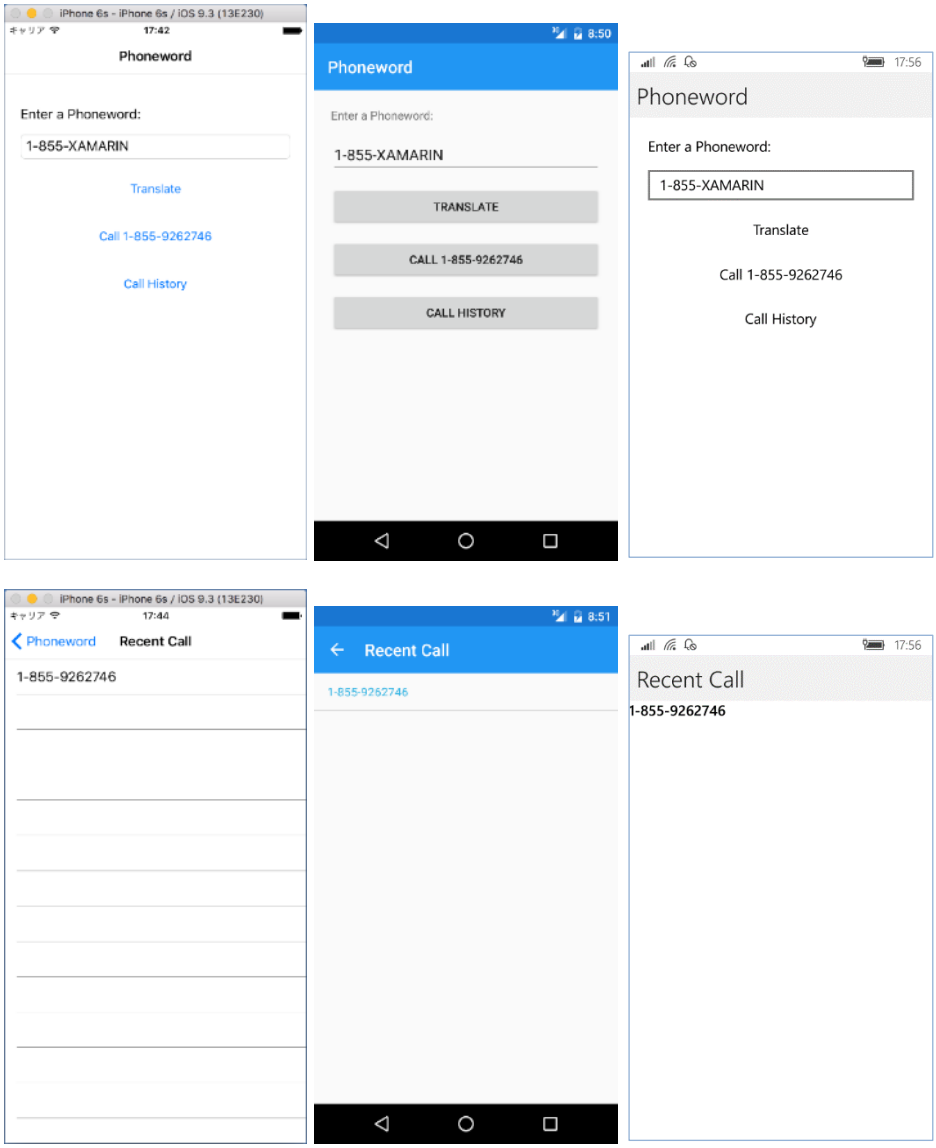
実機へのデプロイは Apple Developer Program（ https://developer.apple.com/programs/jp/ ）に加入するか、Free Provisioning を利用する方法があります。

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。

1. Mac OS が動作している PC と iOS 端末を USB ケーブルで接続します。
2. Visual Studio 上部のメニューバーにある、3 つのコンボボックスを次のように設定します。
   - ✓ ソリューション構成：**Debug**
   - ✓ ソリューション プラットフォーム：**iPhone**
   - ✓ スタートアップ プロジェクト： **Phoneword.iOS**
3. 次に、デバッグ ターゲットから接続した iOS 端末名を選択し、デバッグを開始します。

演習 2: Xamarin.Forms MultiScreen Quick start

この演習では、演習 1 で作成したアプリケーションに、発信履歴機能を追加します。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。



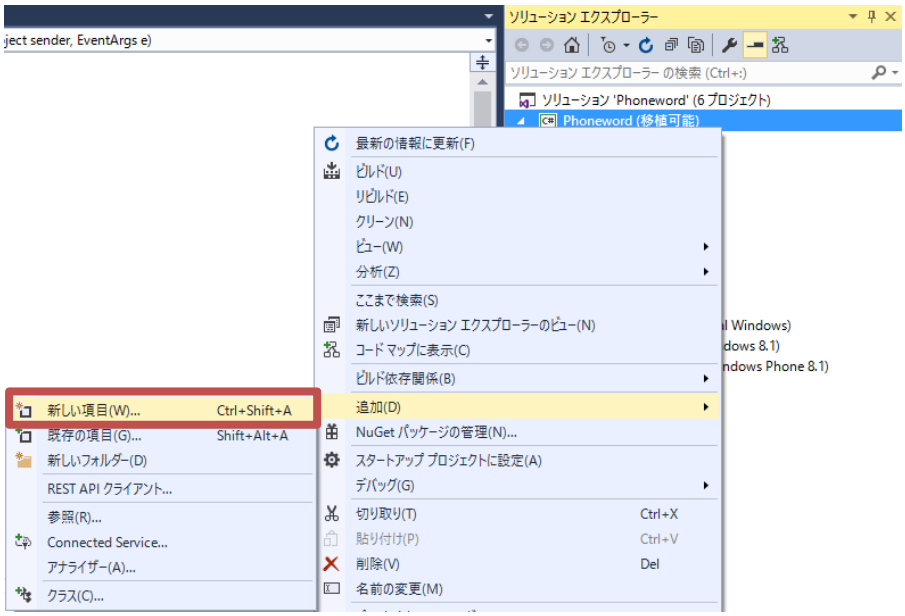
タスク 1 – プロジェクトを開く

1. Visual Studio を起動し、演習 1 で利用した Phoneword プロジェクトを開きます。

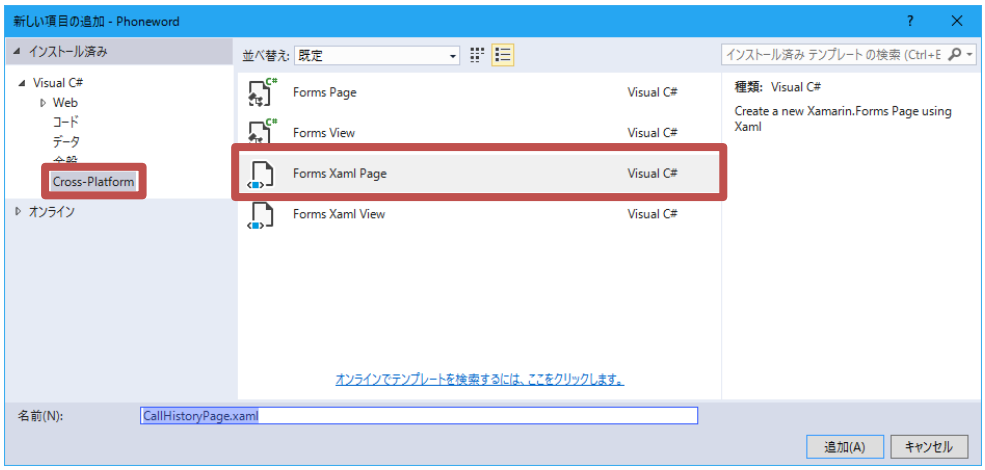
タスク 2 – 発信履歴画面の作成

発信履歴画面を新しく作成します。ListView を用いて、複数の電話番号が表示できるようします。

1. Phoneword プロジェクトを右クリックし、[追加 > 新しい項目] を選択します。



2. [新しい項目の追加] ウィンドウで、左ペインから [Visual C# > Cross-Platform] を選択します。
[Forms Xaml Page] を選択し、名前を [CallHistoryPage.xaml] と入力して、[追加] ボタンをクリックします。



3. 開かれた [CallHistoryPage.xaml] の内容をすべて削除し、以下のコードで置き換えます。

更新前

CallHistoryPage.xaml

```
<?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.CallHistoryPage">
   <Label Text="{Binding MainText}" VerticalOptions="Center"
HorizontalOptions="Center" />
</ContentPage>
```

更新後

CallHistoryPage.xaml

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage
    x:Class="Phoneword.CallHistoryPage"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:Phoneword;assembly=Phoneword">
    <ContentPage.Padding>
        <OnPlatform
            x:TypeArguments="Thickness"
            Android="20, 20, 20, 20"
            WinPhone="20, 20, 20, 20"
            iOS="20, 40, 20, 20" />
    </ContentPage.Padding>
    <ContentPage.Content>
        <StackLayout
            HorizontalOptions="FillAndExpand"
```

```
            Orientation="Vertical"
            Spacing="15"
            VerticalOptions="FillAndExpand">
            <ListView ItemsSource="{x:Static local:App.PhoneNumbers}" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Ctrl+S キーを押して、保存をしてください。

タスク 3 – 発信履歴データの保持

発信履歴のデータを App クラスで保持し、発信履歴画面で表示するために、プロパティで公開を行います。

1. [ソリューション エクスプローラー] から [App.xaml.cs] を開きます。下記のようにコードを修正することで、[PhoneNumbers] プロパティを用意に発信履歴データを保存します。

更新前

App.xaml.cs

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

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

[assembly:XamlCompilation(XamlCompilationOptions.Compile)]

namespace Phoneword
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            MainPage = new Phoneword.MainPage();
        }
        // ・・・・省略・・・・
    }
}
```

更新後

App.xaml.cs

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

```
using System.Linq;
using System.Text;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

[assembly:XamlCompilation(XamlCompilationOptions.Compile)]

namespace Phoneword
{
    public partial class App : Application
    {
        public static IList<string> PhoneNumbers { get; set; }
        public App()
        {
            InitializeComponent();
            PhoneNumbers = new List<string>();
            MainPage = new Phoneword.MainPage();
        }
        // ・・・・省略・・・・
    }
}
```

Ctrl+S キーを押して、保存をしてください。

タスク 4 – 画面遷移に対応する

Xamarin.Forms では、複数のページ間の遷移を行うために、NavigationPage というクラスがあります。初期起動時に NavigationPage を表示し、NavigationPage の中に 表示するページを渡すことで、画面遷移が可能なアプリケーションの実装が可能です。

1. [ソリューション エクスプローラー] から [App.xaml.cs] を開きます。下記のように、App クラスの MainPage プロパティに、コンストラクタの引数に既存のメイン画面を渡した、NavigationPage を設定します。

   更新前

   App.xaml.cs

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

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

[assembly:XamlCompilation(XamlCompilationOptions.Compile)]

namespace Phoneword
```

```
{
    public partial class App : Application
    {
        public static IList<string> PhoneNumbers { get; set; }
        public App()
        {
            InitializeComponent();
            PhoneNumbers = new List<string>();
            MainPage = new Phoneword.MainPage();
        }
        // ・・・・省略・・・・
    }
}
```

更新後

App.xaml.cs

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

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

[assembly:XamlCompilation(XamlCompilationOptions.Compile)]

namespace Phoneword
{
    public partial class App : Application
    {
        public static IList<string> PhoneNumbers { get; set; }
        public App()
        {
            InitializeComponent();
            PhoneNumbers = new List<string>();
            MainPage = new NavigationPage( new Phoneword.MainPage());
        }
        // ・・・・省略・・・・
    }
}
```

Ctrl+S キーを押して、保存をしてください。

タスク 5 – 発信履歴画面を表示する機能を追加する

続いて、発信履歴画面を表示するためのボタンを設置し、App クラスに実装した PhoneNumbers に発信した電話番号を追加する処理を追加します。

1. [ソリューション エクスプローラー] から [MainPage.xaml] を開きます。下記のようにボタンを追加します。

更新前

MainPage.xaml

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage
    x:Class="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="20, 20, 20, 20"
            WinPhone="20, 20, 20, 20"
            iOS="20, 40, 20, 20" />
    </ContentPage.Padding>
    <ContentPage.Content>
        <StackLayout
            HorizontalOptions="FillAndExpand"
            Orientation="Vertical"
            Spacing="15"
            VerticalOptions="FillAndExpand">
            <Label Text="Enter a Phoneword:" />
            <Entry x:Name="phoneNumberText" Text="1-855-XAMARIN" />
            <Button
                x:Name="translateButon"
                Clicked="OnTranslate"
                Text="Translate" />
            <Button
                x:Name="callButton"
                Clicked="OnCall"
                IsEnabled="false"
                Text="Call" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

更新後

MainPage.xaml

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage
    x:Class="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="20, 20, 20, 20"
            WinPhone="20, 20, 20, 20"
            iOS="20, 40, 20, 20" />
    </ContentPage.Padding>
    <ContentPage.Content>
        <StackLayout
            HorizontalOptions="FillAndExpand"
            Orientation="Vertical"
            Spacing="15"
            VerticalOptions="FillAndExpand">
            <Label Text="Enter a Phoneword:" />
            <Entry x:Name="phoneNumberText" Text="1-855-XAMARIN" />
            <Button
                x:Name="translateButon"
                Clicked="OnTranslate"
                Text="Translate" />
            <Button
                x:Name="callButton"
                Clicked="OnCall"
                IsEnabled="false"
                Text="Call" />
            <Button
                x:Name="callHistoryButton"
                Clicked="OnCallHistory"
                IsEnabled="false"
                Text="Call History" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Ctrl+S キーを押して、保存をしてください。

2. [ソリューション エクスプローラー] から [MainPage.xaml.cs] を開きます。下記のようにボタンが押された時の画面遷移と、発信後に PhoneNumbers プロパティに電話番号を追加する処理を記述します。

更新前

MainPage.xaml.cs

```
using System;
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);
            }
        }
    }
}
```

更新後

MainPage.xaml

```
using System;
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)
                {
                    App.PhoneNumbers.Add(translatedNumber);
                    callHistoryButton.IsEnabled = true;
                    dialer.Dial(translatedNumber);
                }
            }
        }

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

Ctrl+S キーを押して、保存をしてください。

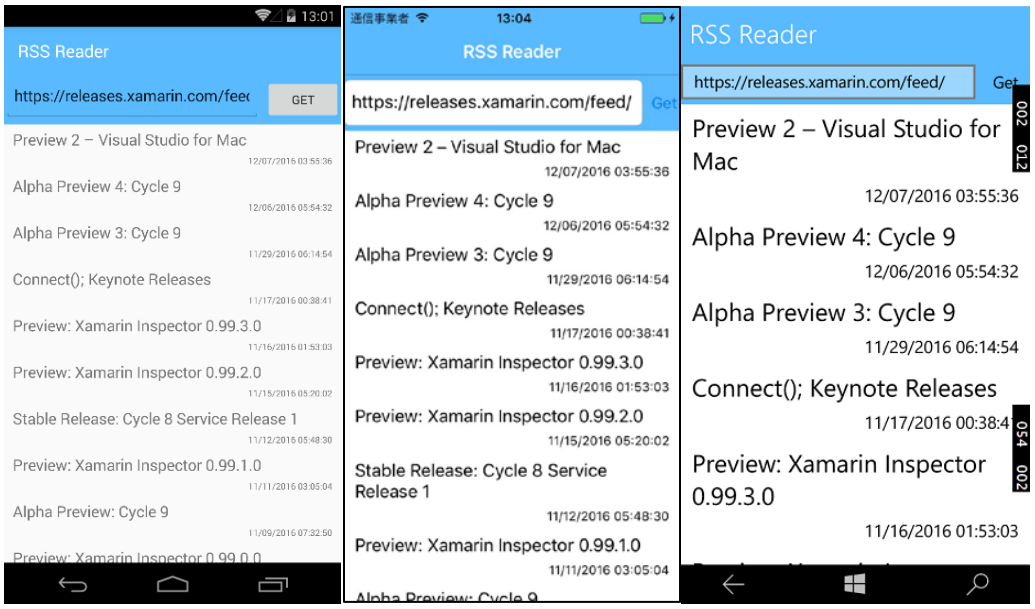
タスク 6 – アプリケーションのデバッグ実行

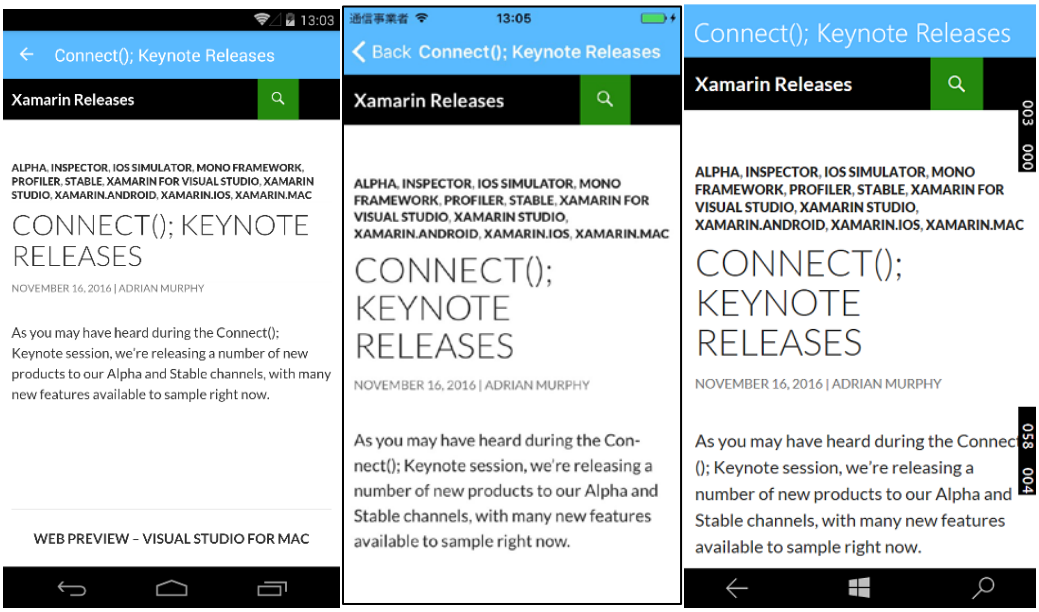
1. [演習 1 、タスク 9 – アプリケーションのデバッグ実行] を参考にして、各プラットフォームでデバッグ実行し、動作を確認します。

演習 3: MVVM パターンを用いた、RSS リーダーアプリの作成

このセクションでは、Xamarin.Forms を使用して、MVVM パターンを用いて RSS リーダーを作成する方法を説明します。

このアプリケーションが完成すると、以下のようになります。

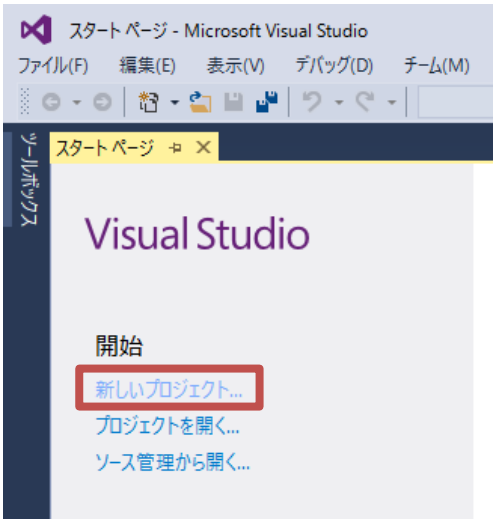




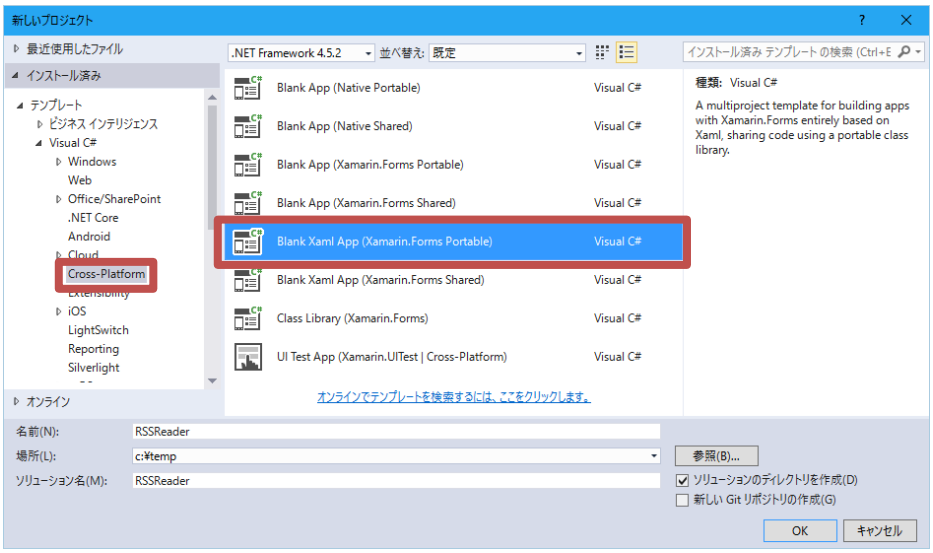
MVVM とは Model-View-ViewModel の略で、WPF / Silverlight における 実装パターンであり、ドメイン ロジックとプレゼンテーション ロジックを分離することを目的とした ユーザー インターフェイス パターンです。現在では、Xaml 形式 で記述できる UWP / Xamarin Forms などもこのパターンで実装することが一般的となりつつあります。

タスク 1 – プロジェクトの新規作成

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



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

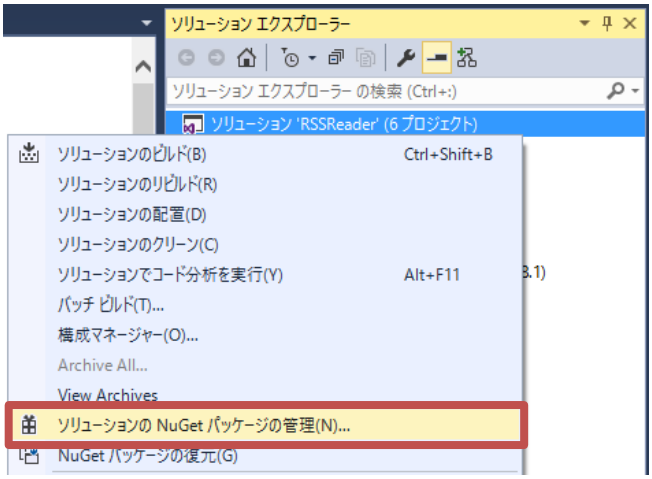


3. プロジェクト作成中に、UWP プロジェクトのバージョンを指定するウィンドウが表示されますが、そのまま [OK] ボタンをクリックします。

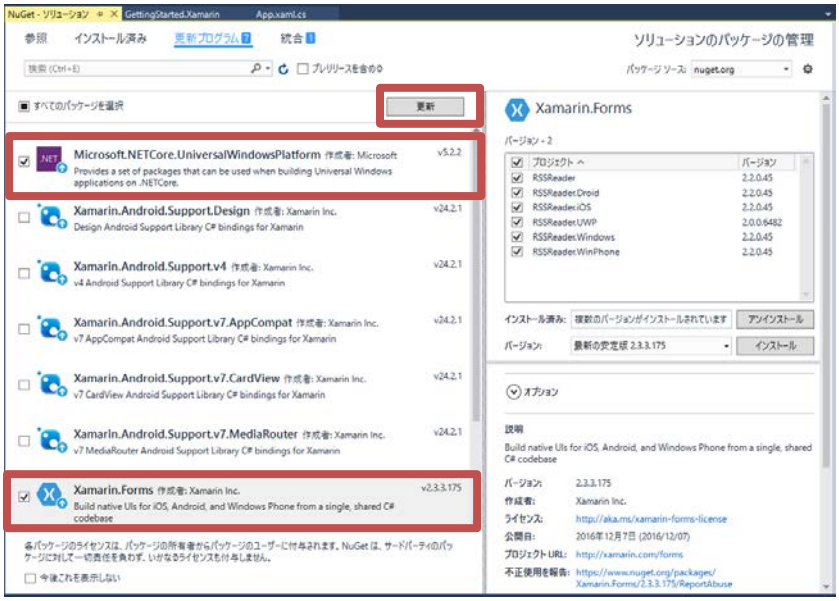
タスク 2 – NuGet パッケージの更新

全てのプロジェクトで必要な NuGet パッケージはソリューション テンプレートにてインストール済みですが、テンプレート作成時点の古いバージョンになっています。そこで、利用している NuGet パッケージを最新バージョンに更新を行います。

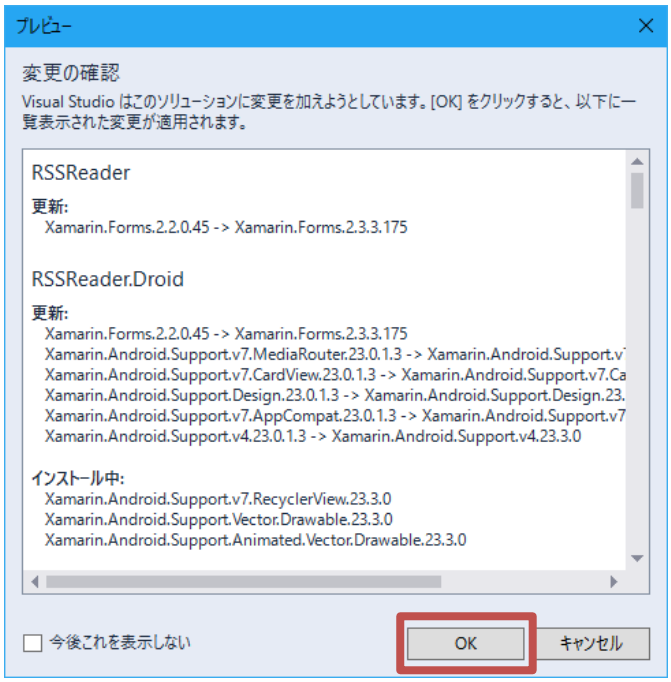
1. パッケージの更新は、[ソリューション エクスプローラー] より [RSSReader] ソリューションを右クリックし、[ソリューションの NuGet パッケージの管理] をクリックします。



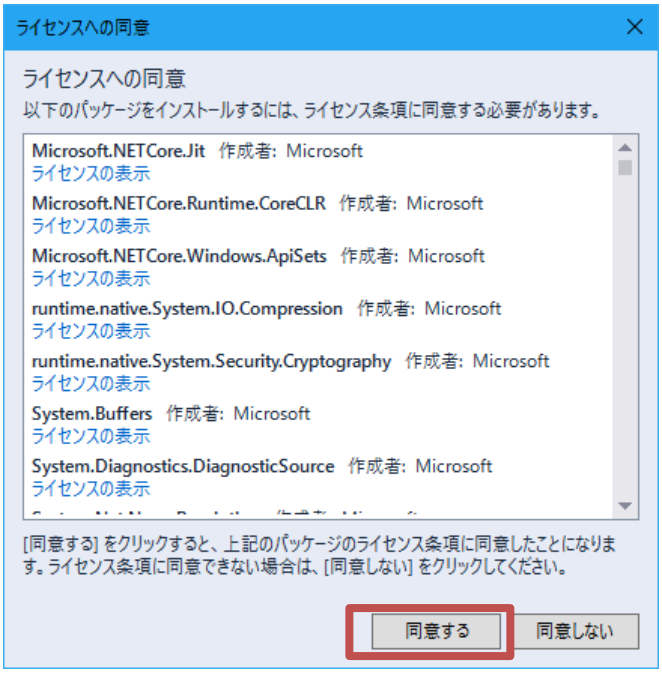
2. [NuGet ソリューションのパッケージの管理] ダイアログで [更新プログラム] タブをクリックし、**[Microsoft.NETCore.UniversalWindowsPlatform] と [Xamarin.Forms] にチェックをして**、[更新] ボタンをクリックします。



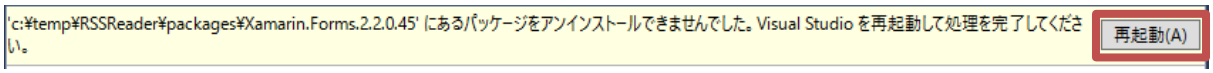
3. [更新の確認] ダイアログが表示されますので、[OK] ボタンをクリックします。



4. [ライセンスへの同意] ダイアログが表示されるので、内容を確認し、[同意する] ボタンをクリックします。



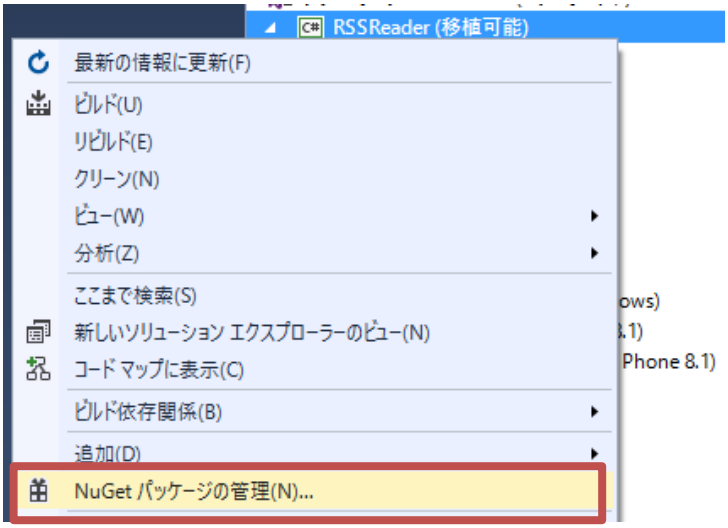
5. [NuGet ソリューションのパッケージの管理] ダイアログの上部に、下記のような表示が出た場合は、[再起動] ボタンをクリックして、Visual Studio を再起動します。



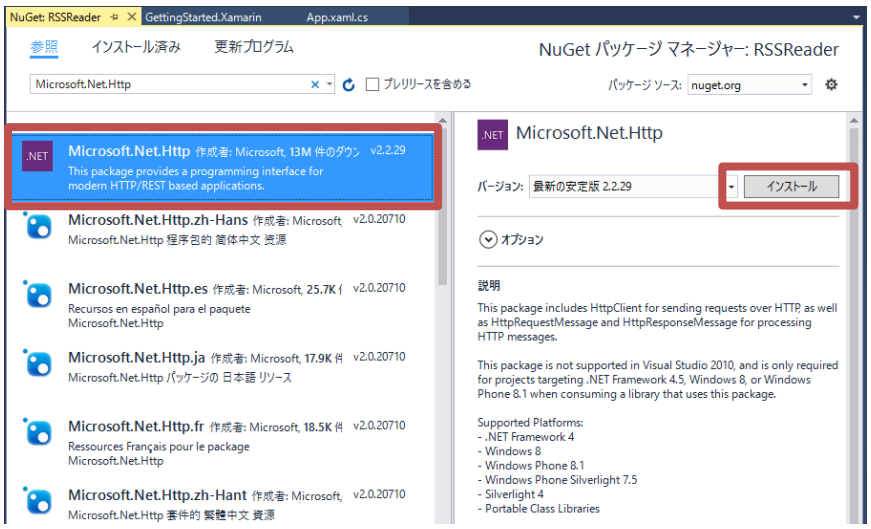
タスク 3 – Model の作成

続いて、RSS Feed を取得し Feed クラスに格納する、Model を担当する実装を行います。

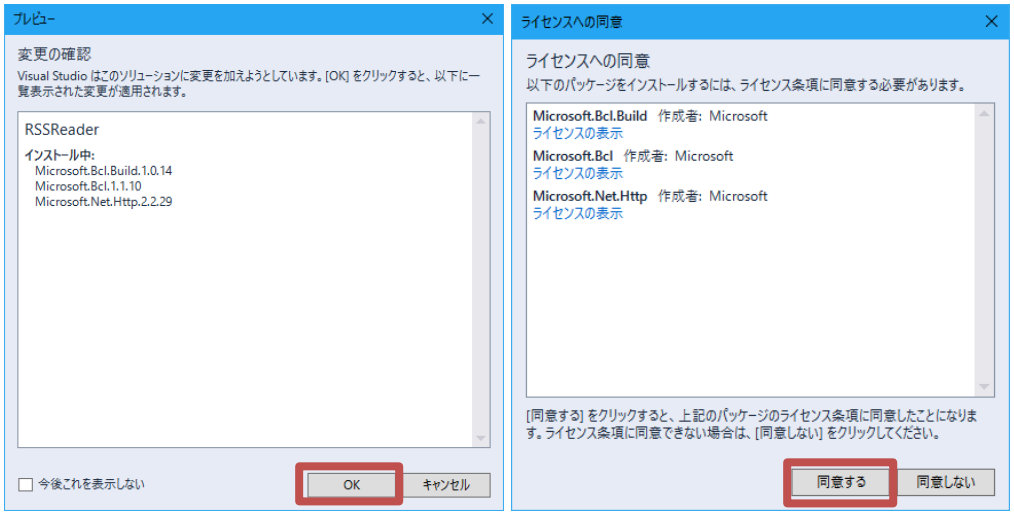
1. インターネット上から RSS Feed を取得するため、HttpClient のライブラリを Nuget から取得します。ソリューション エクスプローラーで、[RSSReader (移植可能)] プロジェクトを右クリックし、[NuGet パッケージの管理] をクリックします。



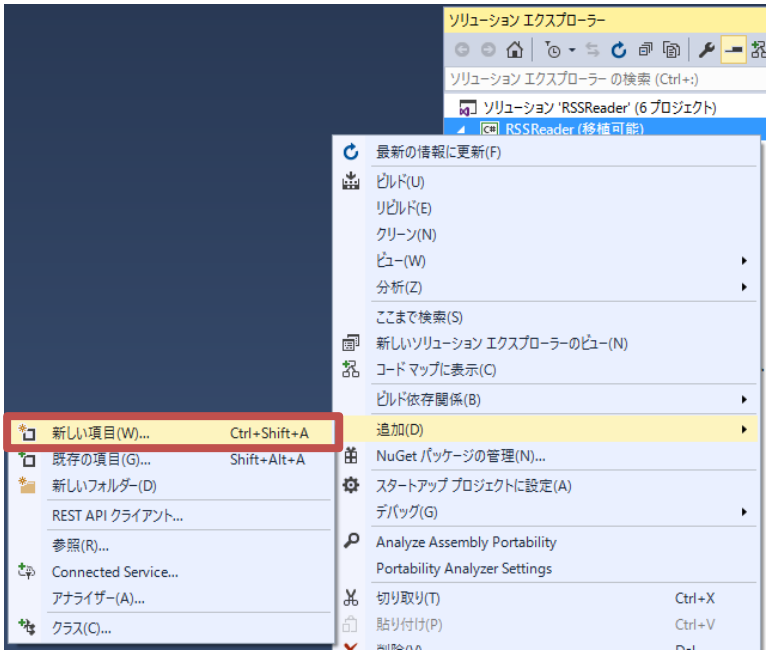
2. NuGet パッケージ マネージャー ウィンドウで、[参照] タブをクリックし、[Microsoft.Net.Http] で検索を行います。表示された、[Microsoft.Net.Http] パッケージを選択し、[インストール] ボタンを押します。



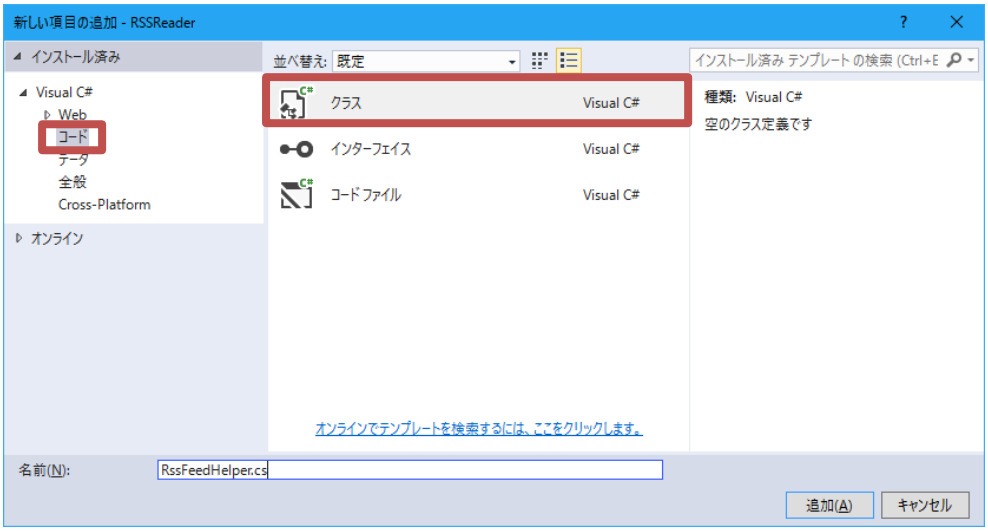
3. [プレビュー] ウィンドウ、[ライセンスへの同意] ウィンドウが表示されますので、それぞれ内容を確認し、[OK] 、[同意する] をクリックし、インストールを行います。



4. 次に、Model となる RSS Feed をインターネットから取得するクラスを作成します。ソリューション エクスプローラーで、[RSSReader (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。



5. [新しい項目の追加] ウィンドウから、[Visual C# > コード > クラス] を選択し、新しいファイルの名前を [RssFeedHelper.cs] と付け、[追加] ボタンをクリックします。



6. RssFeedHelper.cs の内容を、次のコードに置き換えます。

更新前

RssFeedHelper.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RSSReader
```

```
{
    class RssFeedHelper
    {
    }
}
```

更新後

RssFeedHelper.cs

```
using System;
using System.Collections.ObjectModel;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace RSSReader
{
    /// <summary>
    /// RSS Feed を解析するクラス
    /// </summary>
    public static class RSSFeedHelper
    {
        /// <summary>
        /// RSS Feed を取得し、Feed クラスに格納して返却します
        /// </summary>
        /// <param name="feedUri">取得する URL</param>
        /// <returns>取得したフィード</returns>
        public static async Task<ObservableCollection<Feed>> GetRssFeed(Uri
feedUri)
        {
            // NuGet から Microsoft.Net.Http を取得することで、PCL 内で
HttpClient が使えるようになる
            HttpClient client = new HttpClient();

            // 非同期なので await キーワードを付与している、メソッドには async も
            string feedsString = await client.GetStringAsync(feedUri);

            XDocument doc = XDocument.Parse(feedsString);
            var feeds = (from item in
doc.Element("rss").Element("channel").Elements("item")
                         select new Feed()
                         {
                             Title = item.Element("title").Value,
                             Link = item.Element("link").Value,
                             Description = item.Element("description").Value,
                             PublicationDate =
DateTime.Parse(item.Element("pubDate").Value),
                             GUID = item.Element("guid").Value
                         });
```

```
            return new ObservableCollection<Feed>(feeds);
        }
    }
}
```

Ctrl+S キーを押して、保存をしてください。

7. 先ほどと同様に、Feed クラスを作成し、下記のコードに置き換えます。

更新前

Feed.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RSSReader
{
    class Feed
    {
    }
}
```

更新後

Feed.cs

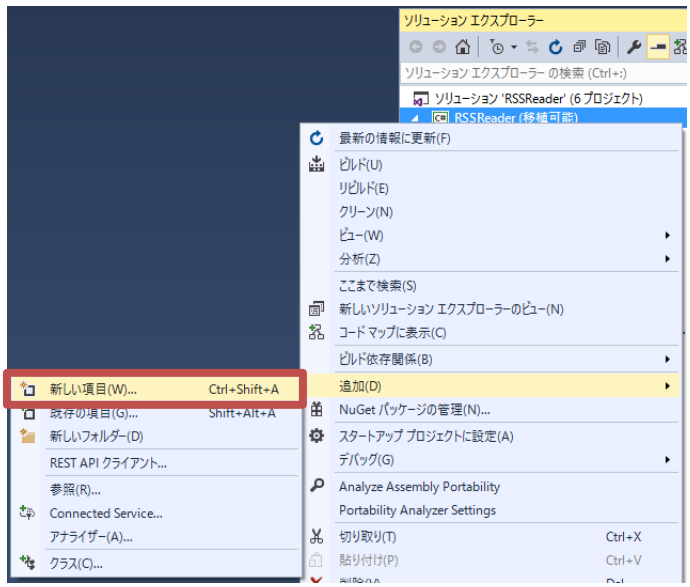
```
using System;

namespace RSSReader
{
    public class Feed
    {
        public string Title { get; set; }
        public string Link { get; set; }
        public string Description { get; set; }
        public DateTime PublicationDate { get; set; }
        public string GUID { get; set; }
    }
}
```

Ctrl+S キーを押して、保存をしてください。

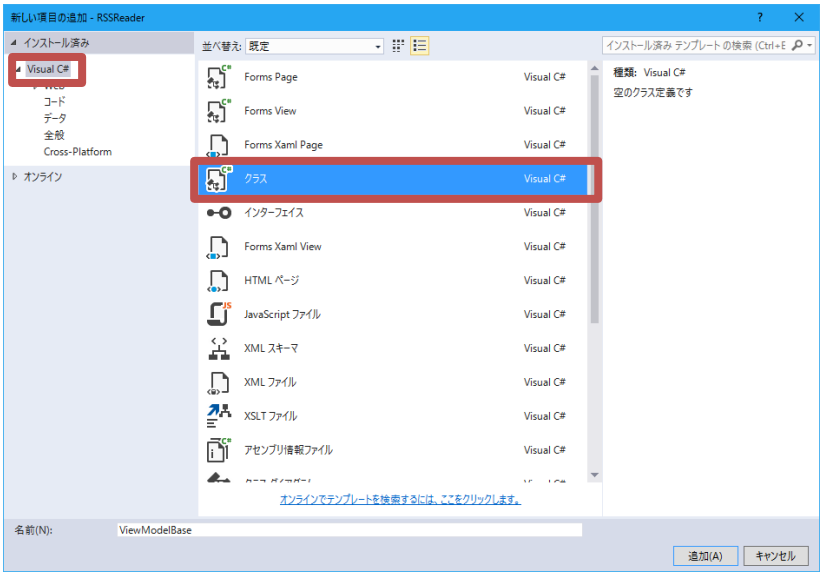
タスク 4 – ViewModel の作成

続いて、View と Model の橋渡し役となる、ViewModel となるクラスを作成します。

1. [ソリューション エクスプローラー] で、[RSSReader (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. [新しい項目の追加] ウィンドウから、[Visual C# > クラス] を選択し、新しいファイルの名前を ViewModelBase と付け、[追加] ボタンをクリックします。



3. ViewModelBase.cs ファイルの内容を、下記のコードに置き換えます。この後作成する、ViewModel クラスの基底クラスとなります。

更新前

ViewModelBase.cs

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

```
using System.Text;
using System.Threading.Tasks;

namespace RSSReader
{
    class ViewModelBase
    {
    }
}
```

更新後

ViewModelBase.cs

```
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using Xamarin.Forms;

namespace RSSReader
{
    public abstract class ViewModelBase : INotifyPropertyChanged
    {
        public INavigation Navigation { get; set; }

        public event PropertyChangedEventHandler PropertyChanged;

        protected void OnPropertyChanged([CallerMemberName]string propertyName
= "")
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Ctrl+S キーを押して、保存をしてください。

4. 先ほどと同様に ViewModel クラスを新規作成し、下記のコードに置き換えます。

更新前

ViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RSSReader
{
```

```
    class ViewModel
    {
    }
}
```

更新後

ViewModel.cs

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

namespace RSSReader
{
    public class ViewModel : ViewModelBase
    {
        public ViewModel() : base()
        {
            Url = "https://releases.xamarin.com/feed/";
        }

        private ObservableCollection<Feed> _feeds;

        public ObservableCollection<Feed> Feeds
        {
            get { return _feeds; }
            set
            {
                _feeds = value;
                OnPropertyChanged();
            }
        }

        private string _url;

        public string Url
        {
            get { return _url; }
            set
            {
                _url = value;
                OnPropertyChanged();
                GetCommand.ChangeCanExecute();
            }
        }

        private Feed _selectedItem;

        public Feed SelectedItem
        {
            get { return _selectedItem; }
            set
```

```
            {
                _selectedItem = value;
                OnPropertyChanged();
            }
        }

        private bool _isBusy = false;

        public bool IsBusy
        {
            get { return _isBusy; }
            set
            {
                _isBusy = value;
                OnPropertyChanged();
            }
        }

        public Command GetCommand
        {
            get
            {
                return new Command(
                   async _ =>
                   {
                       IsBusy = true;
                       Feeds = await RSSFeedHelper.GetRssFeed(new
Uri(this.Url));
                       IsBusy = false;
                   },
                    _ => !string.IsNullOrWhiteSpace(Url));
            }
        }
    }
}
```

Ctrl+S キーを押して、保存をしてください。

これで、View への Data Binding 用の各プロパティ、ボタンを押したとき動作となる Command などが実装されました。

タスク 5 – View の作成

1. View を作成します。MainPage.xaml を開き、下記のコードに置き換えます。

更新前

MainPage.xaml

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

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:RSSReader"
             x:Class="RSSReader.MainPage">

   <Label Text="Welcome to Xamarin Forms!"
          VerticalOptions="Center"
          HorizontalOptions="Center" />

</ContentPage>
```

更新後

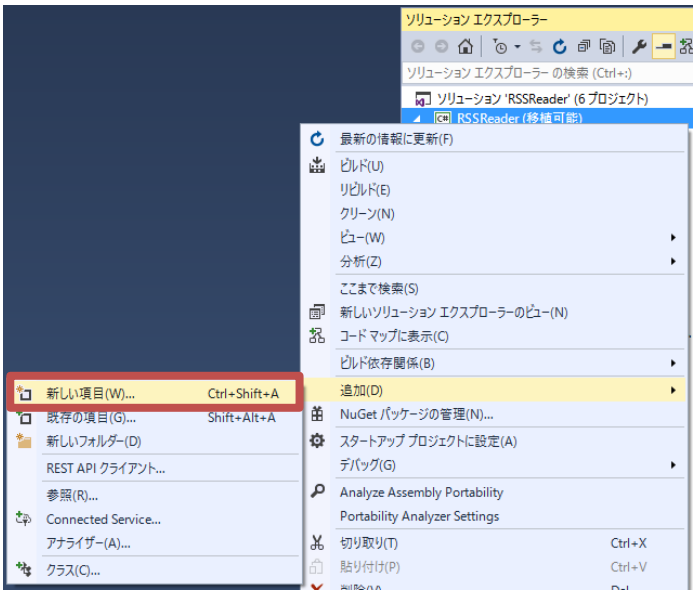
MainPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    x:Class="RSSReader.MainPage"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:RSSReader"
    Title="RSS Reader">
    <Grid RowSpacing="0">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Grid BackgroundColor="#5ABAFF" Padding="0,5">
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition Width="Auto"/>
            </Grid.ColumnDefinitions>
            <!--  URL 入力用の Entry  -->
            <Entry Text="{Binding Url}" />
            <!--  Feed 取得ボタン  -->
            <Button Grid.Column="1" Command="{Binding GetCommand}" Text="Get"
/>
        </Grid>
        <!--  取得した Feed を表示する ListView、縦方向に残った部分に Expand して
いる  -->
        <!--  SeparatorVisibility は Row 毎の線を表示するかどうか、HasUnevenRows
は Row 毎に動的に高さを変更するかどうか  -->
        <ListView
            Grid.Row="1"
            HasUnevenRows="True"
            ItemsSource="{Binding Feeds}"
            SelectedItem="{Binding SelectedItem}"
            SeparatorVisibility="None"
            VerticalOptions="FillAndExpand">
            <ListView.ItemTemplate>
                <!--  各アイテムのテンプレート  -->
                <DataTemplate>
                    <!--  ViewCell は 独自にテンプレートを構築できる  -->
```

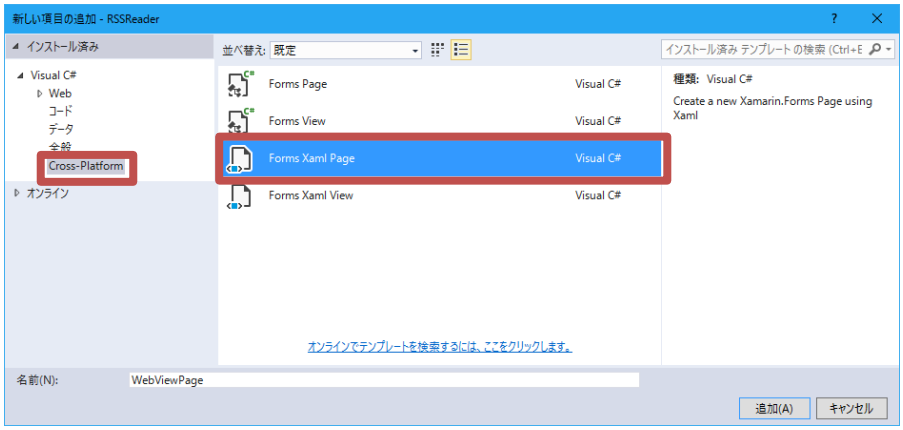
```
                    <ViewCell>
                        <StackLayout Padding="10,5">
                            <!--  各 Cellの BindingContext に一つ一つの Feed オブジェクトがセットされるので、Feed の各プロパティで Binding 構文が書ける  -->
                            <Label
                                FontSize="Medium"
                                HorizontalOptions="StartAndExpand"
                                LineBreakMode="WordWrap"
                                Text="{Binding Title}" />
                            <Label
                                FontSize="Micro"
                                HorizontalOptions="End"
                                Text="{Binding PublicationDate}" />
                        </StackLayout>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
        <ActivityIndicator
            Grid.RowSpan="2"
            IsRunning="{Binding IsBusy}"
            IsVisible="{Binding IsBusy}"
            VerticalOptions="Center" />
    </Grid>
</ContentPage>
```

Ctrl+S キーを押して、保存をしてください。

2. 選択された Feed の Url を表示するために、Web ページを表示するための Page を作成します。
3. ソリューション エクスプローラーで、[RSSReader (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。



1. [新しい項目の追加] ウィンドウから、[Visual C# > Cross-Platform > Forms Xaml Page] を選択し、新しいファイルの名前を [WebViewPage] と付け、[追加] ボタンをクリックします。



2. WebViewPage.xaml の内容を、下記のコードに置き換えます。

更新前

WebViewPage.xaml

```
<?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="RSSReader.WebViewPage">
   <Label Text="{Binding MainText}" VerticalOptions="Center"
HorizontalOptions="Center" />
</ContentPage>
```

更新後

WebViewPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage x:Class="RSSReader.WebViewPage"
             xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             Title="{Binding SelectedItem.Title}">
    <!--  WebView で Web サイトを表示する  -->
    <WebView x:Name="webView" Source="{Binding SelectedItem.Link}" />
</ContentPage>
```

Ctrl+S キーを押して、保存をしてください。

タスク 6 – View と ViewModel の接続

View と ViewModel が作成できたので、それぞれを Data Binding で接続するための実装を行います。

1. まずは、App クラスに ViewModel を保持させ、また、ナビゲーションが可能になるように、NavigationPage を利用する実装に変更します。App.xaml.cs ファイルを開き、namespace の追加と、Xaml のコンパイル指定を追加します。

更新前

WebViewPage.xaml

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

using Xamarin.Forms;

namespace RSSReader
{
    public partial class App : Application
    {
```

更新後

WebViewPage.xaml

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

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

[assembly:XamlCompilation(XamlCompilationOptions.Compile)]

namespace RSSReader
{
    public partial class App : Application
    {
```

2. 続いて、下記の太字の部分を書き換えます。

更新前

WebViewPage.xaml

```
public partial class App : Application
{
    public App()
    {
        InitializeComponent();
```

```
        MainPage = new RSSReader.MainPage();
    }
```

更新後

WebViewPage.xaml

```
public partial class App : Application
{
    // ViewModel を Singleton で保持
    public static ViewModel ViewModel { get; private set; } = new ViewModel();

    public App()
   {
        InitializeComponent();

        // ナビゲーションを実現するために、NavigationPage を生成して、その最初のページとして MainPageを指定
        MainPage = new NavigationPage(new RSSReader.MainPage())
        {
            BarBackgroundColor = Color.FromHex("#5ABAFF"),
            BarTextColor = Color.White
        };
        // ナビゲーションを ViewModel で実行するために、 INavigation を渡している
        ViewModel.Navigation = MainPage.Navigation;
    }
```

Ctrl+S キーを押して、保存をしてください。

3. MainPage.xaml.cs を開き、BindingContext プロパティに先ほどの ViewModel を設定するコードを追加します。

更新前

MainPage.xaml.cs

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }
}
```

更新後

MainPage.xaml.cs

```
public partial class MainPage : ContentPage
{
```

```
    public MainPage()
    {
        InitializeComponent();

        // ViewModel を Binding できるように設定
        BindingContext = App.ViewModel;
    }
}
```

この実装で、View と ViewModel の Data Binding が接続されます。
Ctrl+S キーを押して、保存をしてください。

4. 同様に、WebViewPage.xaml.cs を開き、BindingContext に先ほどの ViewModel を設定するコードを追加します。

更新前

WebViewPage.xaml.cs

```
public partial class WebViewPage : ContentPage
{
    public WebViewPage()
    {
        InitializeComponent();
    }
}
```

更新後

WebViewPage.xaml.cs

```
public partial class WebViewPage : ContentPage
{
    public WebViewPage()
    {
        InitializeComponent();

        // ViewModel を Binding できるように設定
        BindingContext = App.ViewModel;
    }
}
```

Ctrl+S キーを押して、保存をしてください。

5. ViewModel.cs を開き、SelectedItem プロパティで、WebViewPage へのナビゲーション を行うコードを実装します。

更新前

ViewModel.cs

```
public Feed SelectedItem
{
    get { return _selectedItem; }
    set
    {
        _selectedItem = value;
        OnPropertyChanged();
    }
}
```

更新後

ViewModel.cs

```
public Feed SelectedItem
{
    get { return _selectedItem; }
    set
    {
        _selectedItem = value;
        OnPropertyChanged();
        Navigation?.PushAsync(new WebViewPage());
    }
}
```

Ctrl+S キーを押して、保存をしてください。

タスク 7 – アプリケーションのデバッグ実行

1. [演習 1 、タスク 9 – アプリケーションのデバッグ実行] を参考にして、各プラットフォームでデバッグ実行し、動作を確認します。

付録

iOS アプリを Windows 上の Visual Studio からビルド、デバッグ実行するためのセットアップ

ここでは、Visual Studio から Mac OS をネットワーク経由で接続し、iOS アプリケーションのビルド、デバッグをするためのセットアップ手順をご案内いたします。

Mac のセットアップ

Mac OS が動作している 端末をご用意ください。要件は以下の通りです。

- ✓ 最新の iOS SDK (Xcode と同時にインストールされます)
- ✓ 最新の Xcode
- ✓ Mac OS X Yosemite (10.10) またはそれ以上のバージョン

Apple ID の作成

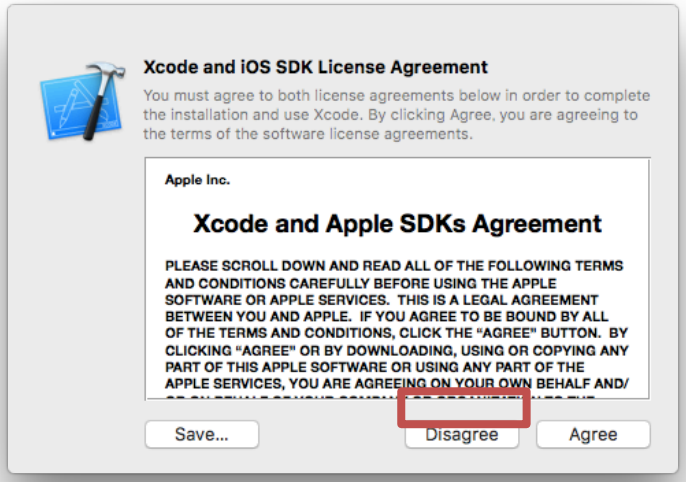
https://appleid.apple.com で無料の Apple ID を作成します (まだ作成していない場合)。これは Xcode をインストールしてサインインするために必要です。

Xcode のダウンロードとインストール

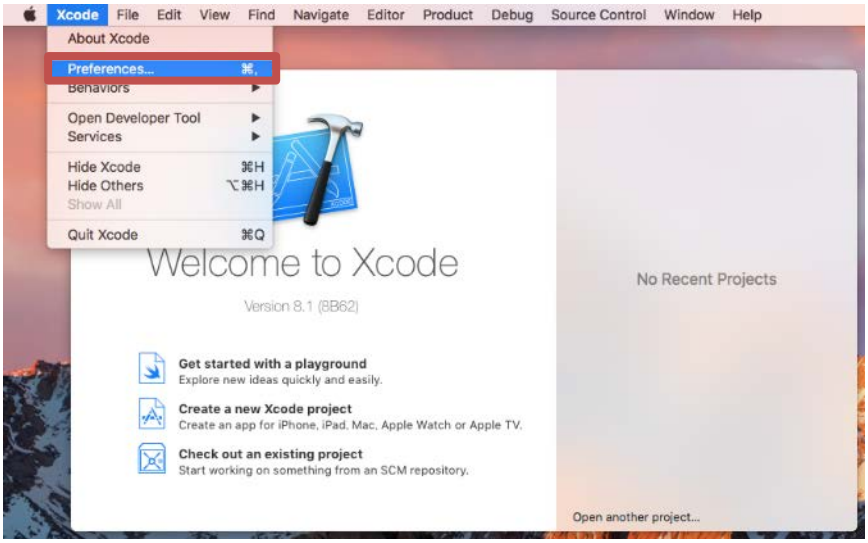
続いて、[App Store] または、https://developer.apple.com/xcode/ から Xcode をダウンロードしてインストールします。

Xcode の初期設定

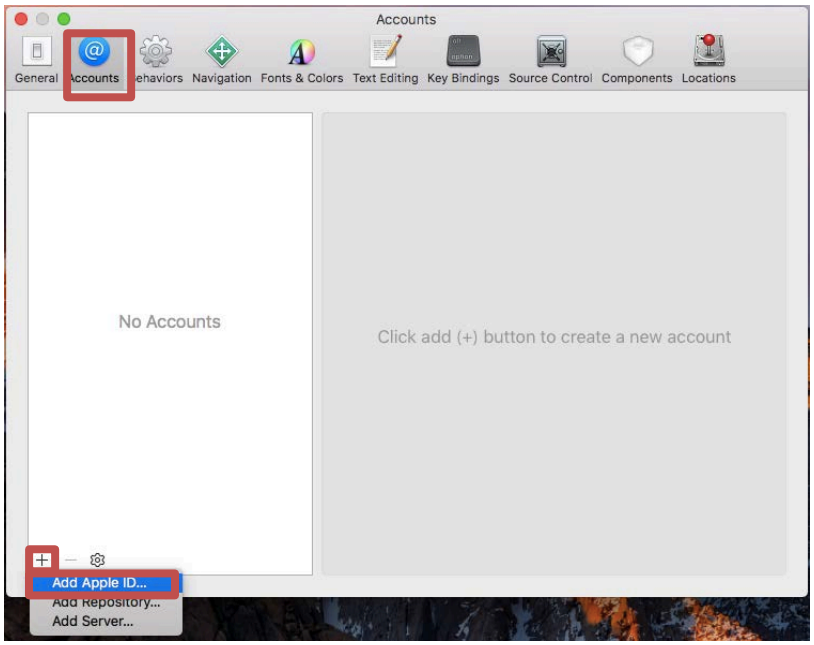
Xcode のインストールが終わりましたら、Xcode を起動します。初めて起動すると、下記のように Lisence Agreement が表示されますので、内容を確認し、[Agree] ボタンをクリックします。



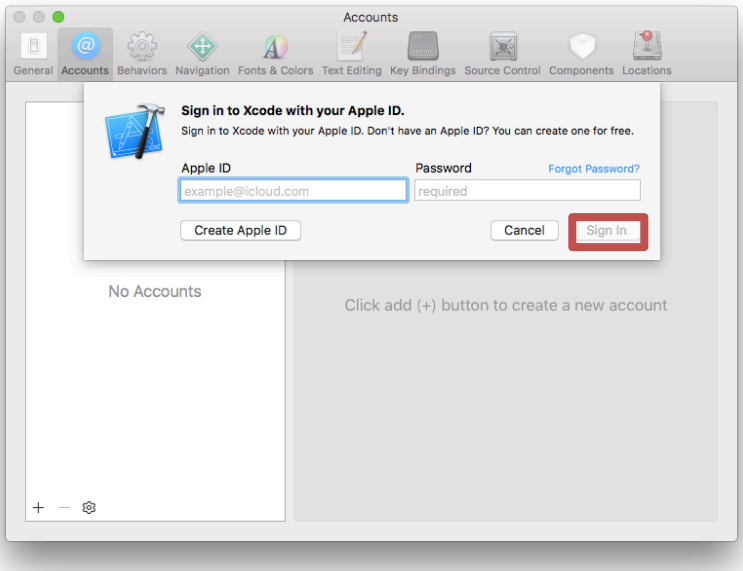
続いて、Xcode に Apple ID を追加します。メニューバーから [Xcode > Preferences] をクリックします。



設定ウィンドウが表示されたら、[Accounts] タブの左ペイン下にある [+] をクリックし、[Add Apple ID...] をクリックします。



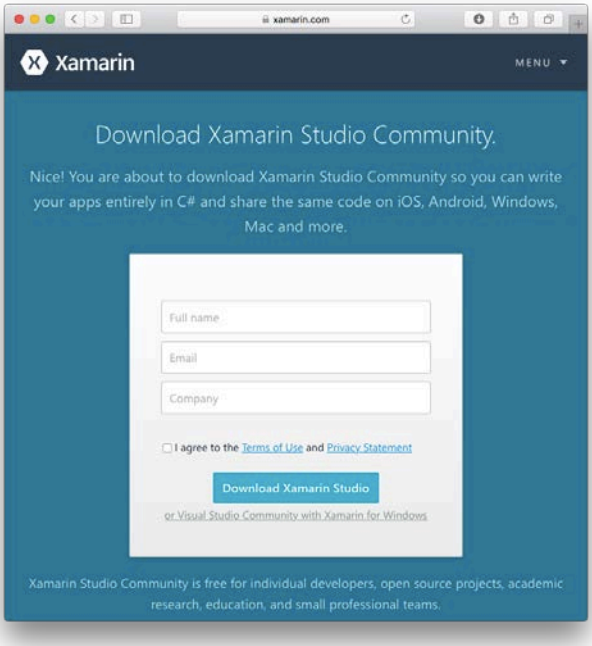
Apple ID とパスワードを入力し、[Sign in] ボタンをクリックします。



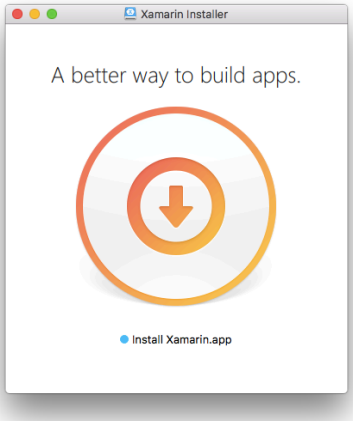
詳細な手順については、「Adding Your Account to XCode (XCode にアカウントを追加する)」(apple.com、英語) をご参照ください。

Xamarin for Mac のダウンロードとインストール

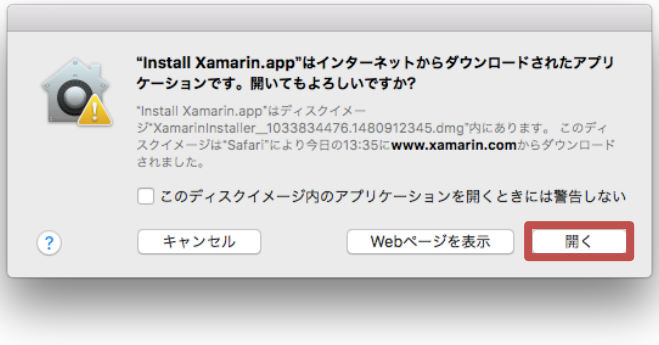
https://xamarin.com/download へアクセスし、Xamarin for Mac をダウンロードします。



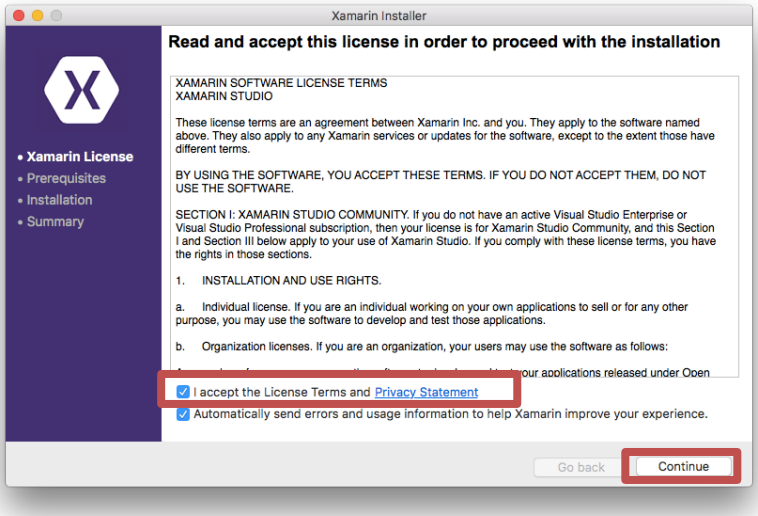
ダウンロードされた *.dmg ファイルをダブルクリックし、Xamarin Installer を起動します。下記のウィンドウが表示されたら、[Install Xamarin.app] をダブルクリックします。



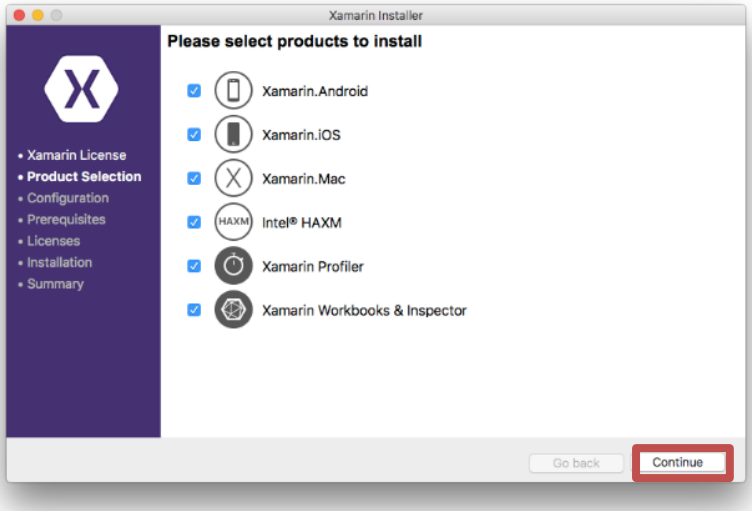
この画面が表示された時は、ダウンロード元を確認し、[開く] ボタンをクリックします。



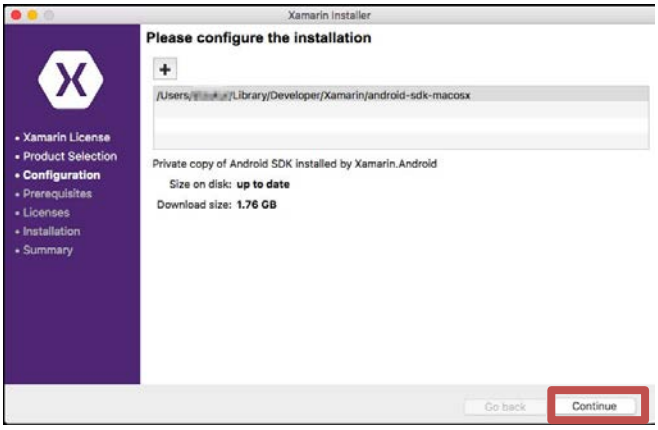
Xamarin のライセンスが表示されますので、内容を確認し、[I accept the License Terms and Privacy Statement] にチェックを入れ、[Continue] をクリックします。



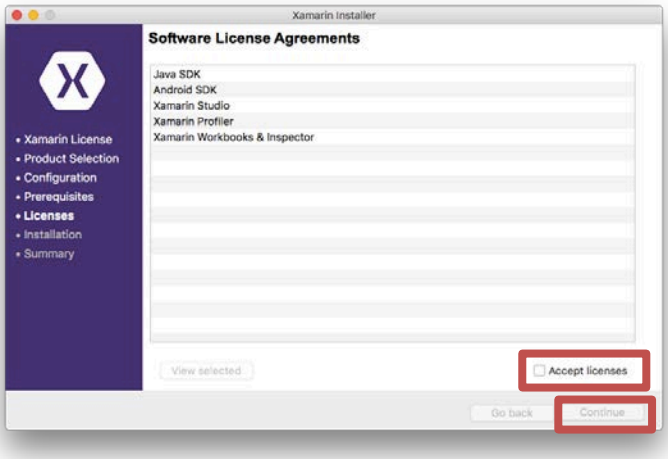
インストールするプロダクトを選択して、[Continue] をクリックします。Visual Studio と接続して iOS のビルド、実行を行うには、[Xamarin.iOS] が必要です。



Xamarin.Android をインストールする場合、Android SDK のインストールフォルダの指定が必要となります。特に問題が無ければ、そのまま [Continue] をクリックしてください。

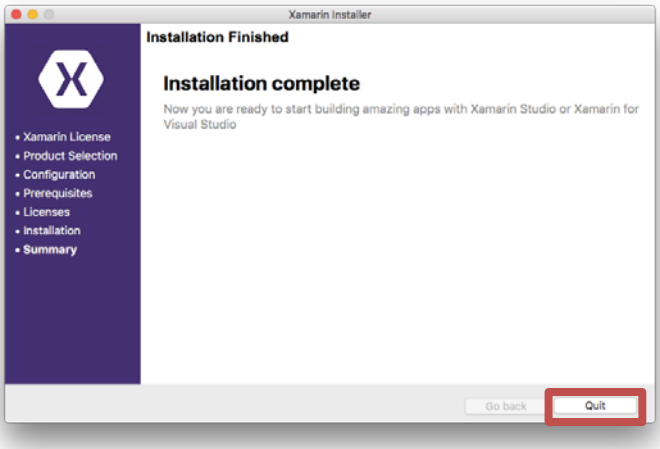


各ソフトウェアの個別の License Agreements が表示されますので、内容を確認し、[Accept licenses] にチェックを入れ、[Continue] をクリックしてください。



ダウンロードとインストールが完了するまで、しばらくお待ちください。

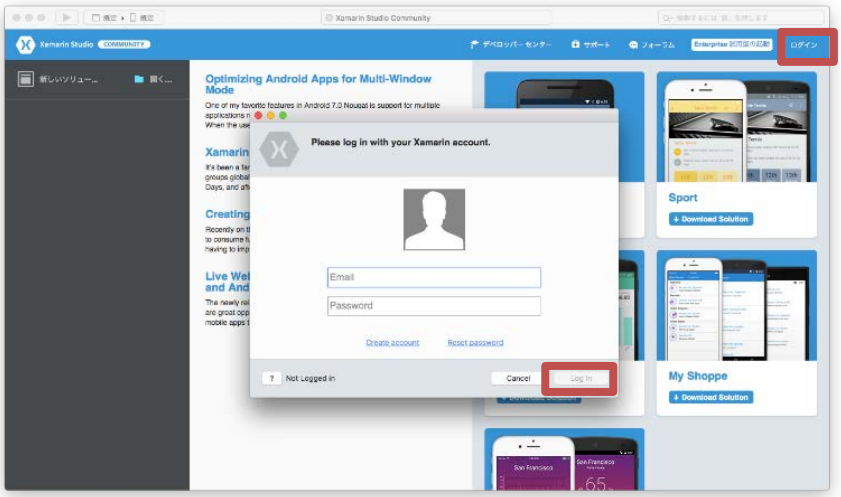
インストールが完了しましたら、[Quit] をクリックして、インストーラーを終了させます。



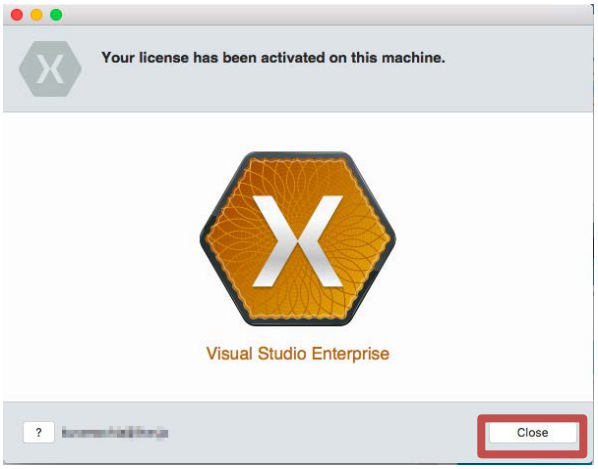
Xamarin アカウントのアクティベーション

Community エディションで利用される方は、この手順は不要です。MSDN サブスクリプション等で、Enterprise エディションをお持ちの方は、「MSDN サブスクリプションのアカウント」より、サブスクリプションのから Xamarin Studio (for OS X) を選択し、登録を行った Xamarin アカウントでアクティベーションを行う必要があります。

Xamarin Studio を起動し、画面右上の [ログイン] をクリックします。表示されたダイアログで Xamarin アカウントを入力し、[Log in] ボタンをクリックします。



アクティベーションに成功すると、下記のように表示されます。(この例は Enterprise エディション) [Close] をクリックし、画面を閉じます。

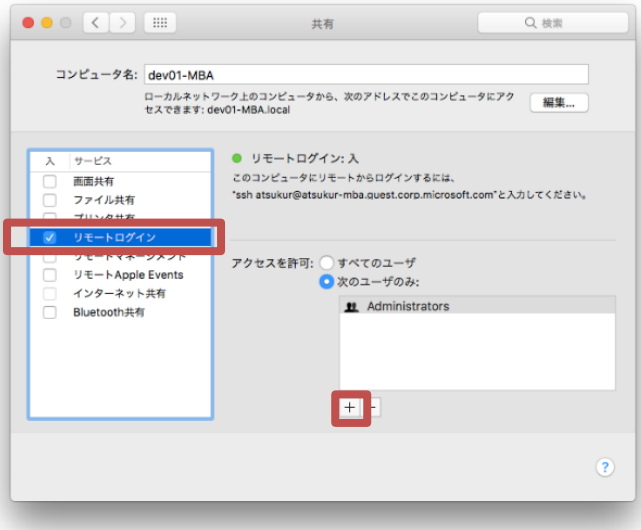


詳細な手順については、「Xamarin.iOS のダウンロードとインストール」(xamarin.com、英語) の手順をご確認ください。

Windows 上の Visual Studio と Mac を接続する

続いて、Windows 上の Visual Studio から、Mac を接続するための設定を行います。まずは、Mac 側の設定として、リモート ログイン を有効にします。Mac のタスクバー右上の リンゴをクリックし、[システム環境設定...] を起動します。下記の画面が表示されたら、[共有] メニューを選択します。

共有ウィンドウの左ペインから [リモート ログイン] にチェックを入れ、[アクセスを許可：] 下部の [＋] ボタンをクリックし、現在のユーザーを追加します。



ネットワーク接続

Visual Studio がインストールされている Windows 端末と、先ほど設定した Mac 端末を、同一ネットワーク上に接続しておきます。

Xamarin の更新

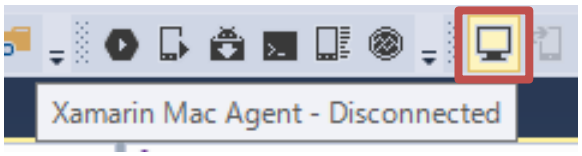
Xamarin を最新のコンポーネントに更新するには、[Xamarin Studio] を起動し、[Xamarin Studio > 更新の確認] メニューを選択します。Windows 上のコンポーネントと Mac 上のコンポーネントが同一バージョンでない場合は、不具合が発生する可能性があります。Windows と Mac は同時にバージョンアップするようにしてください。

Windows 上の Visual Studio との接続

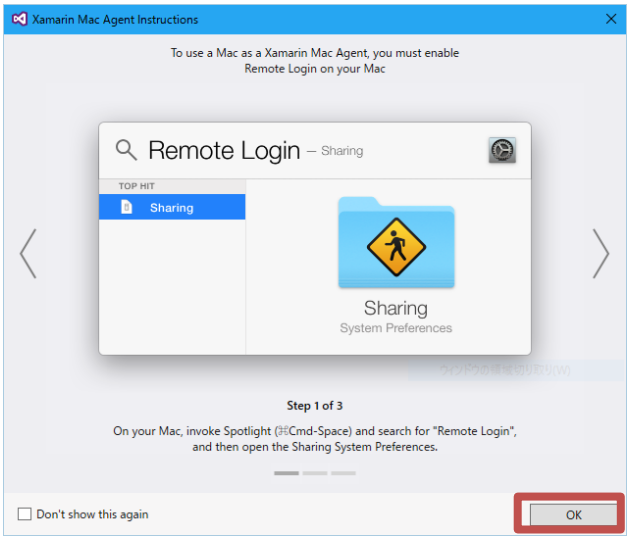
続いて、Windows 上の Visual Studio から、先ほどセットアップを行った Mac 端末との接続の設定を行います。

Windows 端末の Visual Studio の設定

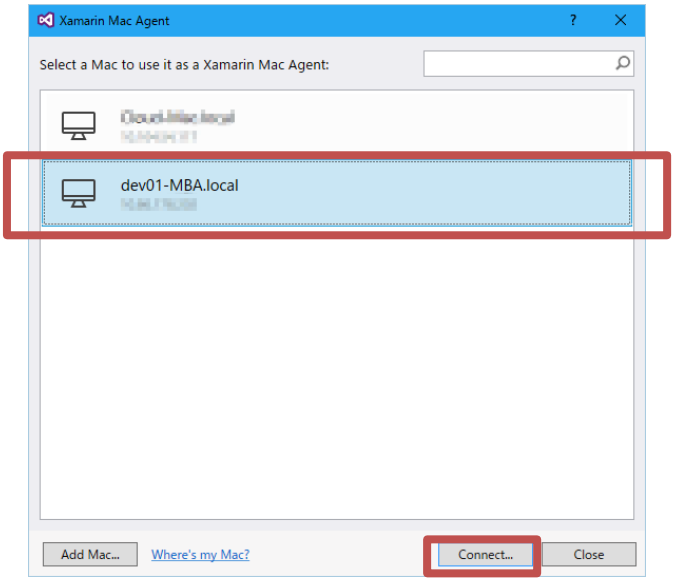
Visual Studio で Xamarin.iOS が含まれているプロジェクト（例えば、演習 1、演習 2 の Phoneword プロジェクト）を開き、ツールバーから、[Xamarin Mac Agent] をクリックします。



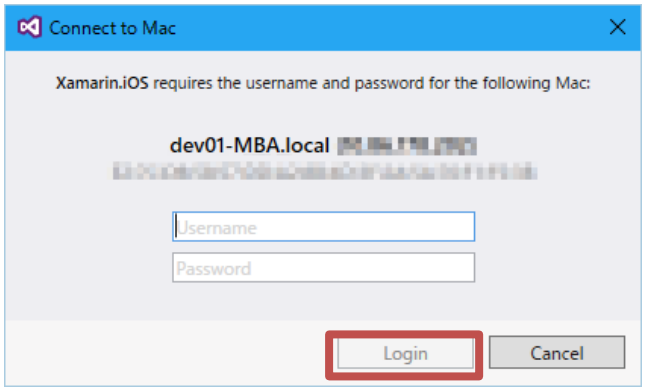
下記のウィンドウが表示されるので、[OK] をクリックします。



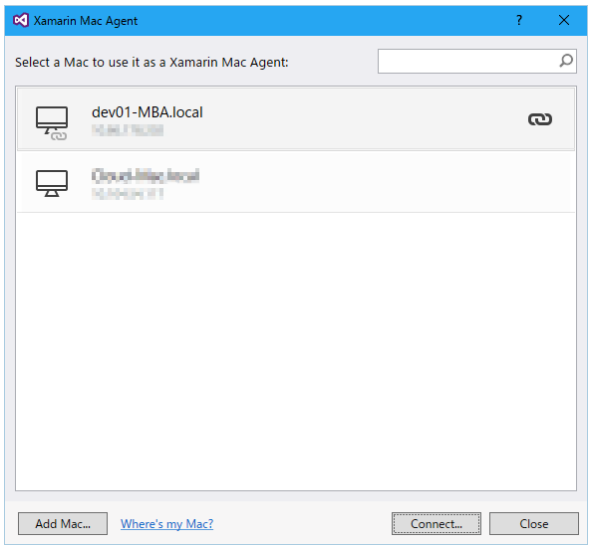
ネットワーク上の接続されている Mac を選択し、[Connect...] ボタンをクリックします。



[Connect to Mac] ウィンドウが表示されたら、Mac 上の ユーザー名とパスワードを入力し、[Login] ボタンをクリックします。



接続が完了すると、鎖マークのアイコンが表示されます。



また、ツールバーの、[Xamarin Mac Agent] が緑色になり、(Connected) と表示されていれば接続済みです。

詳細な手順は「Connecting to the Mac (Mac への接続)」(xamarin.com、英語) を参照ください。

関連リンク

- Visual Studio - Xamarin クロスプラットフォーム開発
  https://www.microsoft.com/ja-jp/dev/campaign/vs-xamarin.aspx
- 公式 Xamarin ドキュメント(英語)
  https://developer.xamarin.com/guides/
- 無料版「Creating Mobile Apps with Xamarin.Forms」電子書籍 (英語)
  https://developer.xamarin.com/guides/xamarin-forms/creating-mobile-apps-xamarin-forms/
- Visual Studio と Xamarin
  https://msdn.microsoft.com/ja-JP/library/mt299001.aspx
- Xamarin Releases
  https://releases.xamarin.com/
- Xamarin Open Source
  http://open.xamarin.com/

- Microsoft エンタープライズ向け Premier サポート
  https://www.microsoft.com/ja-jp/services/premier.aspx


Microsoft