

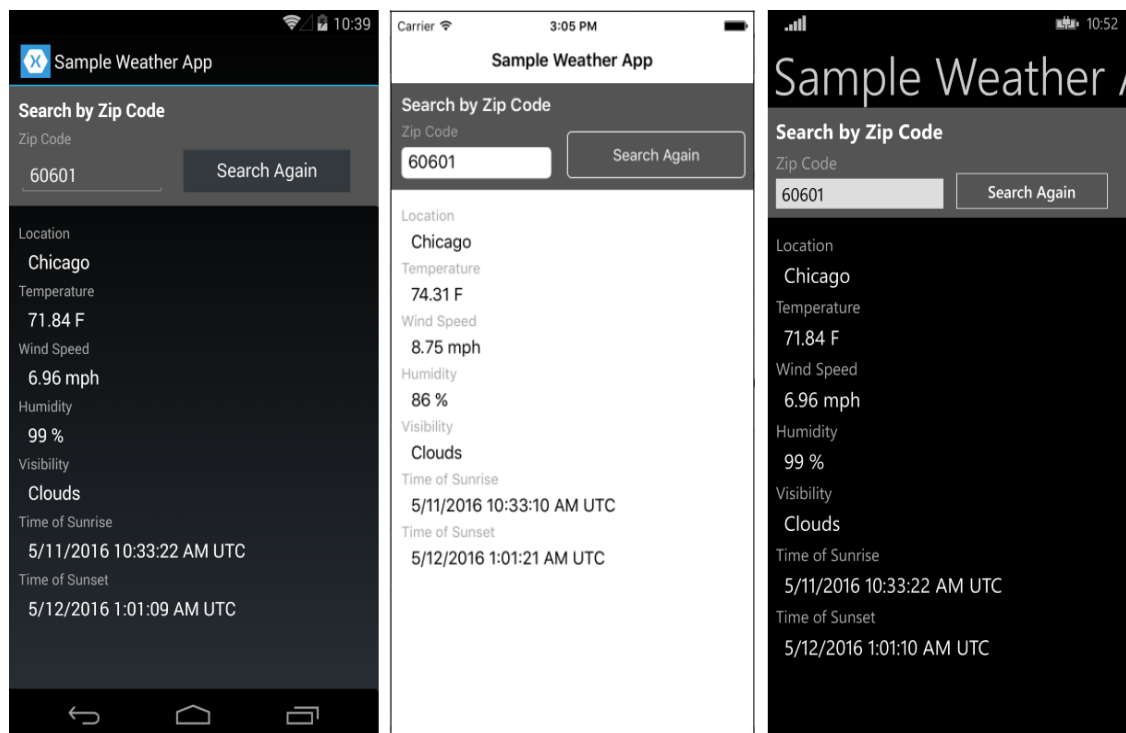
Tutorial : Création d'applications avec Xamarin dans Visual Studio

1 Introduction

Une fois que vous avez configuré, installé et vérifié votre environnement de développement Xamarin, cette procédure pas à pas vous montre comment créer une application de base avec Xamarin.Forms.

Les pages de référence de l'API sont sur : <https://developer.xamarin.com/api/>

Xamarin.Forms vous permet d'écrire tout le code de l'interface utilisateur une seule fois dans une bibliothèque de classes portable (PCL). Xamarin affiche ensuite automatiquement les contrôles de l'interface utilisateur native pour les plateformes iOS, Android et Windows. Nous recommandons cette approche, parce que l'option de bibliothèque PCL prend mieux en charge l'utilisation des seules API .NET qui sont prises en charge sur toutes les plateformes cibles, et parce que Xamarin.Forms vous permet de partager le code de l'interface utilisateur entre les plateformes.



Vous allez effectuer les opérations suivantes pour la générer :

1. Configurer votre solution
2. Écrire le code de service de données partagé
3. Commencer la rédaction du code de l'interface utilisateur partagé

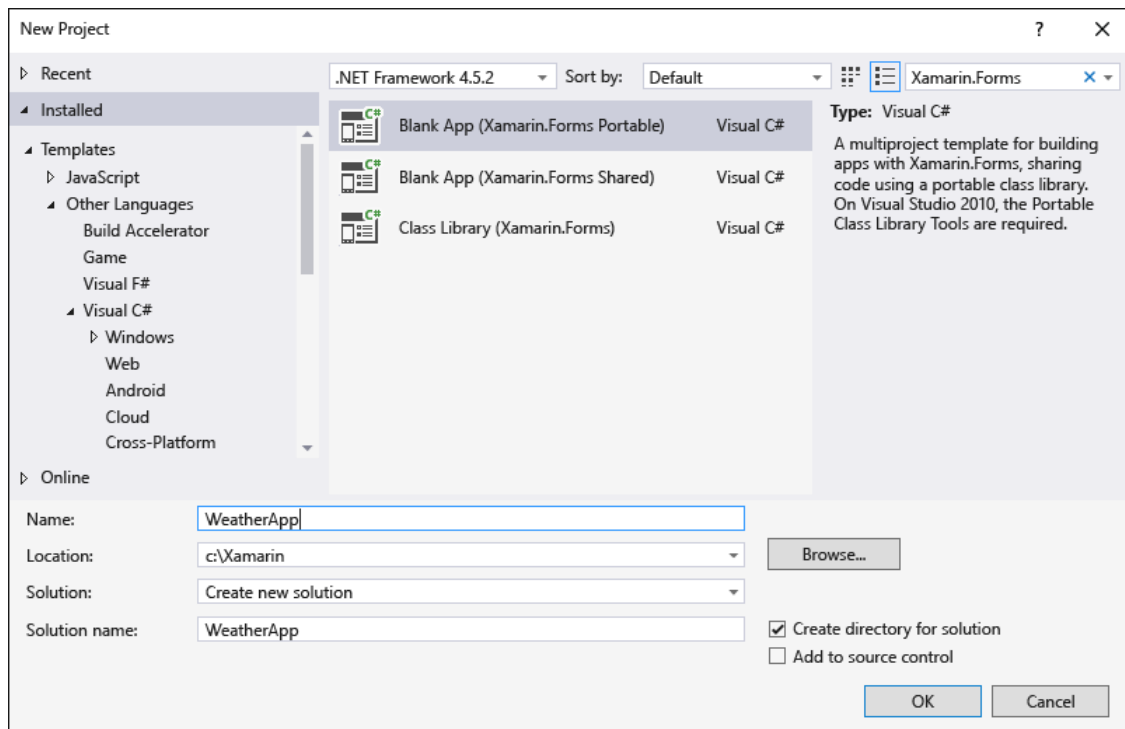
Tutorial : Création d'applications avec Xamarin dans Visual Studio

4. Tester votre application avec l'émulateur Visual Studio pour Android
5. Terminer l'interface utilisateur avec une apparence native entre les plateformes

2 Configurer votre solution

Les étapes suivantes créent une solution Xamarin.Forms qui contient une bibliothèque PCL pour le code partagé et deux packages NuGet ajoutés.

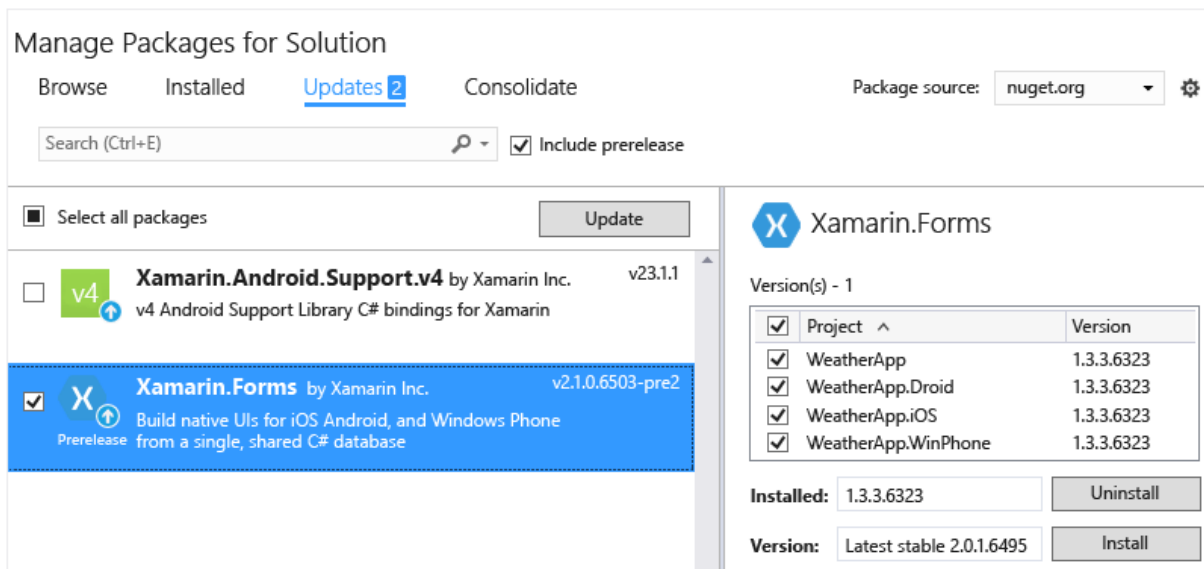
1. Dans Visual Studio, créez une solution **Application vide (Xamarin.Forms Portable)** et nommez-la **WeatherApp**. Vous pouvez trouver plus facilement ce modèle en entrant **Xamarin.Forms** dans le champ de recherche.



2. Après avoir cliqué sur OK pour créer la solution, vous avez un certain nombre de projets individuels :
 - **WeatherApp (Portable)** : bibliothèque PCL où vous allez écrire du code qui est partagé entre les plateformes, notamment une logique métier commune et le code de l'interface utilisateur à l'aide de Xamarin.Forms.
 - **WeatherApp.Droid** : projet qui contient le code Android natif. Ce projet est défini comme projet de démarrage par défaut.
 - **WeatherApp.iOS** : projet qui contient le code iOS natif.
 - **WeatherApp.UWP** : le projet qui contient du code Windows 10 UWP (Universal Windows).

Tutorial : Création d'applications avec Xamarin dans Visual Studio

- **WeatherApp.Windows (Windows 8.1)** : le projet qui contient du code Windows 8.1 natif.
 - **WeatherApp.WinPhone (Windows Phone 8.1)** : le projet qui contient le code Windows Phone natif.
3. Dans chaque projet natif, vous avez accès au concepteur natif pour la plateforme correspondante et pouvez implémenter des écrans et fonctionnalités spécifiques à la plateforme en fonction des besoins.
4. Mettez à niveau le package NuGet Xamarin.Forms dans votre solution vers la dernière version stable comme suit. Nous vous recommandons d'effectuer cette opération chaque fois que vous créez une solution Xamarin :
- Sélectionnez **Outils > Gestionnaire de package NuGet > Gérer les packages NuGet pour la solution**.
 - Sous l'onglet **Mises à jour**, cochez la mise à jour **Xamarin.Forms** et cochez cette option pour mettre à jour tous les projets de votre solution. (Remarque : laissez les mises à jour pour Xamarin.Android.Support décochées.)
 - Mettez à jour le champ **Version** en sélectionnant « **dernière stable** » pour la version disponible.
 - Cliquez sur **Mettre à jour**.



Manage Packages for Solution

Browse Installed **Updates 2** Consolidate Package source: nuget.org

Search (Ctrl+E) Include prerelease

Select all packages Update

Package	Version
Xamarin.Android.Support.v4 by Xamarin Inc.	v23.1.1
Xamarin.Forms by Xamarin Inc.	v2.1.0.6503-pre2

Xamarin.Forms

Version(s) - 1

Project	Version
<input checked="" type="checkbox"/> WeatherApp	1.3.3.6323
<input checked="" type="checkbox"/> WeatherApp.Droid	1.3.3.6323
<input checked="" type="checkbox"/> WeatherApp.iOS	1.3.3.6323
<input checked="" type="checkbox"/> WeatherApp.WinPhone	1.3.3.6323

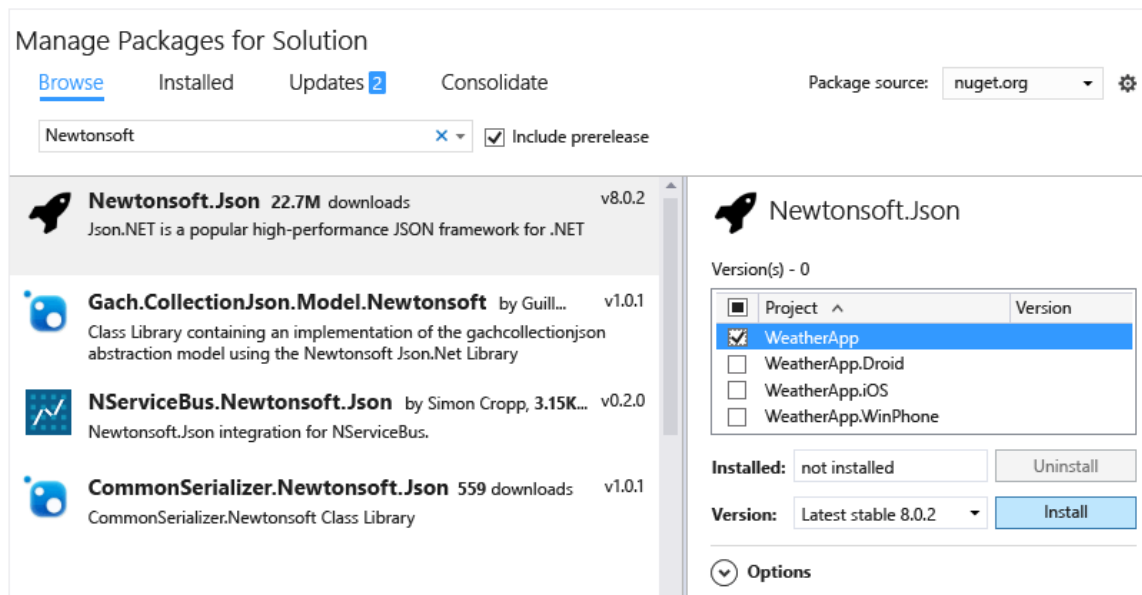
Installed: 1.3.3.6323 Uninstall

Version: Latest stable 2.0.1.6495 Install

5. Ajoutez les packages NuGet et **Newtonsoft.Json** au projet de bibliothèque PCL, que vous allez utiliser pour traiter les informations récupérées à partir d'un service de données météo :
- Dans le Gestionnaire de package NuGet (toujours ouvert depuis l'étape 3), sélectionnez l'onglet **Parcourir** et recherchez **Newtonsoft**.
 - Sélectionnez **Newtonsoft.Json**.
 - Cochez le projet **WeatherApp** (il s'agit du seul projet dans lequel vous devez installer le package).

Tutorial : Création d'applications avec Xamarin dans Visual Studio

- Vérifiez que la valeur définie pour le champ **Version** est **Dernière stable**.
- Cliquez sur **Installer**.



6. Répétez l'étape 4 ci-dessus pour rechercher et installer le package **Microsoft.Net.Http**.
7. Générez votre solution et vérifiez l'absence d'erreur de génération.

3 Écrire le code de service de données partagé

Le projet **WeatherApp (Portable)** est l'emplacement où vous allez écrire du code pour la bibliothèque PCL partagée entre les plateformes. La bibliothèque PCL est automatiquement incluse dans les packages d'application générés par les projets iOS, Android et Windows Phone.

Pour exécuter cet exemple, vous devez tout d'abord vous inscrire pour obtenir une clé API gratuite d'utilisation du web service sur <http://openweathermap.org/appid>.

Les étapes suivantes ajoutent le code à la bibliothèque PCL pour accéder aux données de ce service météo et stocker ces données :

1. Cliquez avec le bouton droit sur le projet **WeatherApp** et sélectionnez **Ajouter > Classe**. Dans la boîte de dialogue **Ajouter un nouvel élément**, nommez le fichier **Weather.cs**. Vous utiliserez cette classe pour stocker les données du service de données météo.
2. Remplacez l'intégralité du contenu de **Weather.cs** par le code suivant :

Tutorial : Création d'applications avec Xamarin dans Visual Studio

```
namespace WeatherApp
{
    public class Weather
    {
        public string Title { get; set; }
        public string Temperature { get; set; }
        public string Wind { get; set; }
        public string Humidity { get; set; }
        public string Visibility { get; set; }
        public string Sunrise { get; set; }
        public string Sunset { get; set; }

        public Weather()
        {
            //Because labels bind to these values, set them to an empty string to
            //ensure that the label appears on all platforms by default.
            this.Title = " ";
            this.Temperature = " ";
            this.Wind = " ";
            this.Humidity = " ";
            this.Visibility = " ";
            this.Sunrise = " ";
            this.Sunset = " ";
        }
    }
}
```

3. Ajoutez une autre classe au projet de bibliothèque PCL nommée **DataService.cs** que vous allez utiliser pour traiter les données JSON du service de données météo.
4. Remplacez l'intégralité du contenu de **DataService.cs** par le code suivant :

```
using System.Threading.Tasks;
using Newtonsoft.Json;
using System.Net.Http;

namespace WeatherApp
{
    public class DataService
    {
        public static async Task<dynamic> getDataFromService(string
queryString)
        {
            HttpClient client = new HttpClient();
            var response = await client.GetAsync(queryString);

            dynamic data = null;
            if (response != null)
            {
                String json =
response.Content.ReadAsStringAsync().Result;
                data = JsonConvert.DeserializeObject(json);
            }
        }
    }
}
```

Tutorial : Création d'applications avec Xamarin dans Visual Studio

```

    }

    return data;
}
}
}

```

- Ajoutez une troisième classe à la bibliothèque PCL nommée **Core**, où vous allez placer la logique métier partagée, telle que la logique qui forme une chaîne de requête avec un code postal, appelle le service de données météo, puis remplit une instance de la classe **Weather**.
- Voici un exemple de code **Core.cs** :

```

using System;
using System.Threading.Tasks;

namespace WeatherApp
{
    public class Core
    {
        public static async Task<Weather> GetWeather(string zipCode)
        {
            //Sign up for a free API key at
            http://openweathermap.org/appid
            string key = "YOUR KEY HERE";
            string queryString =
            "http://api.openweathermap.org/data/2.5/weather?zip="
            + zipCode + ",us&appid=" + key + "&units=imperial";

            dynamic results = await
            DataService.GetDataFromService(queryString).ConfigureAwait(false);

            if (results["weather"] != null)
            {
                Weather weather = new Weather();
                weather.Title = (string)results["name"];
                weather.Temperature = (string)results["main"]["temp"]
                + " F";
                weather.Wind = (string)results["wind"]["speed"] + "
                mph";
                weather.Humidity = (string)results["main"]["humidity"]
                + " %";
                weather.Visibility =
                (string)results["weather"][0]["main"];

                DateTime time = new System.DateTime(1970, 1, 1, 0, 0,
                0, 0);
                DateTime sunrise =
                time.AddSeconds((double)results["sys"]["sunrise"]);
                DateTime sunset =
                time.AddSeconds((double)results["sys"]["sunset"]);
            }
        }
    }
}

```

Tutorial : Création d'applications avec Xamarin dans Visual Studio

```
        weather.Sunrise = sunrise.ToString() + " UTC";  
        weather.Sunset = sunset.ToString() + " UTC";  
        return weather;  
    }  
    else  
    {  
        return null;  
    }  
}  
}
```

7. Générez le projet de bibliothèque PCL **WeatherApp** pour vérifier que le code est correct.

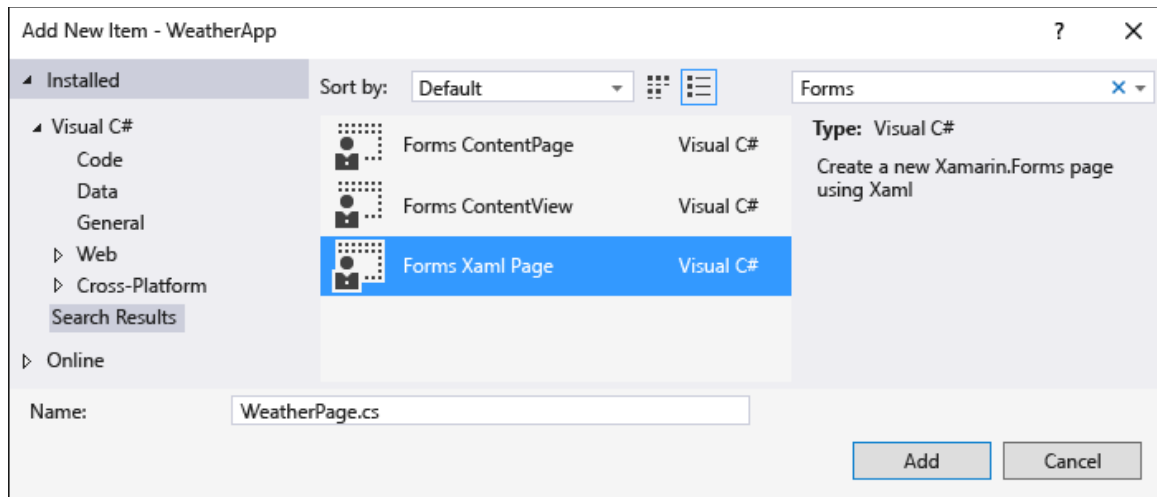
4 Commencer la rédaction du code de l'interface utilisateur partagé

Xamarin.Forms vous permet d'implémenter le code de l'interface utilisateur partagé dans la bibliothèque PCL. Dans cette procédure, vous allez ajouter un écran à la bibliothèque PCL avec un bouton qui met à jour son texte avec les données retournées par le code du service de données météo ajouté dans la section précédente :

1. Ajoutez une **Page XAML Forms** nommée **WeatherPage.cs** en cliquant avec le bouton droit sur le projet **WeatherApp** et en sélectionnant **Ajouter > Nouvel élément...** Dans la boîte de dialogue **Ajouter un nouvel élément**, recherchez « Forms », sélectionnez **Page XAML Forms** et nommez-la **WeatherPage.cs**.

Xamarin.Forms étant basé sur XAML, cette étape crée un fichier **WeatherPage.xaml** ainsi que le fichier code-behind imbriqué **WeatherPage.xaml.cs**. Cela vous permet de générer l'interface utilisateur via XAML ou du code. Vous procéderez de l'une des deux façons dans cette procédure pas à pas.

Tutorial : Création d'applications avec Xamarin dans Visual Studio



2. Pour ajouter un bouton à l'écran WeatherPage, remplacez le contenu de WeatherPage.xaml par le code suivant :

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="WeatherApp.WeatherPage">
  <Button x:Name="getWeatherBtn" Text="Get Weather"/>
</ContentPage>
  
```

Notez que le nom du bouton doit être défini à l'aide de l'attribut **x:Name** pour pouvoir faire référence à ce bouton par son nom depuis le fichier de code associé, appelé aussi « code-behind ».

3. Pour ajouter un gestionnaire d'événements pour l'événement **Clicked** du bouton pour mettre à jour le texte du bouton, remplacez le contenu de **WeatherPage.xaml.cs** par le code ci-dessous. (Vous pouvez remplacer « 60601 » par un autre code postal).

```

using System;
using Xamarin.Forms;

namespace WeatherApp
{
  public partial class WeatherPage: ContentPage
  {
    public WeatherPage ()
    {
      InitializeComponent();
      this.Title = "Sample Weather App";
    }
  }
}
  
```


Tutorial : Création d'applications avec Xamarin dans Visual Studio

```
getWeatherBtn.Clicked += GetWeatherBtn_Clicked;

//Set the default binding to a default object for now
this.BindingContext = new Weather();
}

private async void GetWeatherBtn_Clicked(object sender,
EventArgs e)
{
    Weather weather = await Core.GetWeather("60601");
    getWeatherBtn.Text = weather.Title;
}
}
```

4. Pour ouvrir **WeatherPage** en tant que premier écran quand l'application démarre, remplacez le constructeur par défaut dans **App.cs** par le code suivant :

```
public App()
{
    MainPage = new NavigationPage(new WeatherPage());
}
```

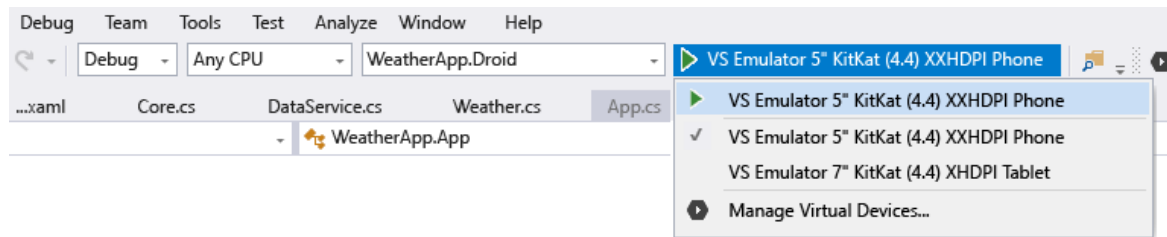
5. Générez le projet de bibliothèque PCL WeatherApp pour vérifier que le code est correct.

5 Tester votre application avec l'émulateur Visual Studio pour Android

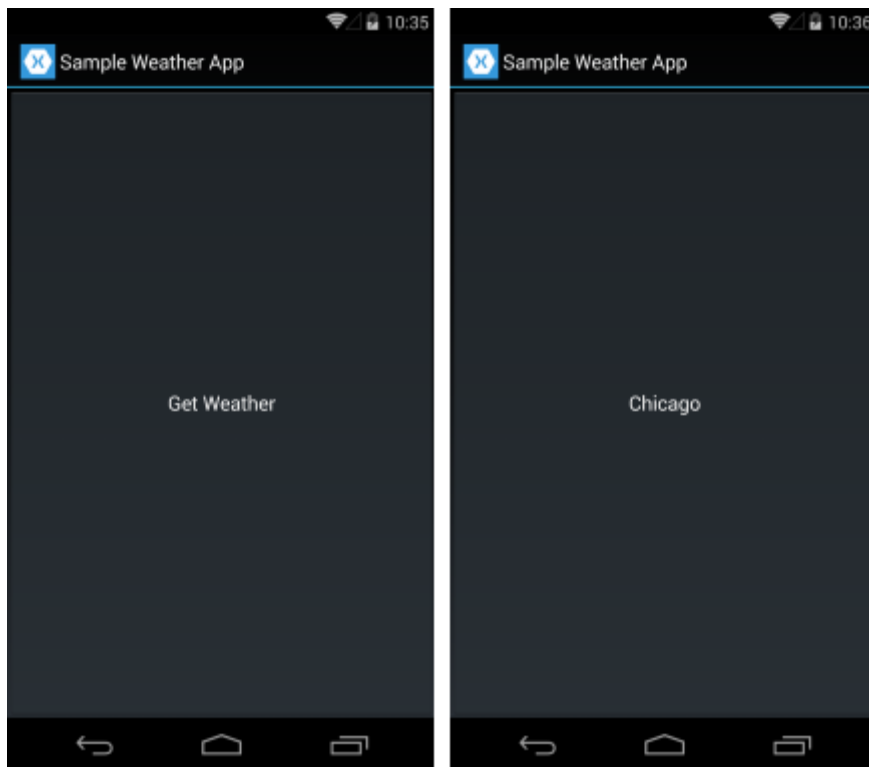
Vous êtes maintenant prêt à exécuter l'application ! À présent, exécutez simplement la version Android pour vérifier que l'application obtient des données du service météo. Par la suite, vous allez également exécuter les versions iOS et Windows Phone après avoir ajouté d'autres éléments d'interface utilisateur. (Remarque : si vous exécutez Visual Studio sous Windows 7, suivez la même procédure, mais avec Xamarin Player à la place.)

1. Définissez **WeatherApp.Droid** comme projet de démarrage en cliquant dessus avec le bouton droit et en sélectionnant **Définir comme projet de démarrage**.
2. Dans la barre d'outils Visual Studio, vous verrez **WeatherApp.Droid** répertorié en tant que projet cible. Sélectionnez l'un des émulateurs Android pour le débogage et appuyez sur **F5**. Nous vous recommandons d'utiliser l'une des options de l'**émulateur Visual Studio** pour l'exécution de l'application dans l'émulateur Visual Studio pour Android.

Tutorial : Création d'applications avec Xamarin dans Visual Studio



3. Quand l'application démarre dans l'émulateur, cliquez sur le bouton **Get Weather**. Le texte du bouton doit être mis à jour et contenir **Chicago, IL**, qui est la propriété *Title* des données récupérées à partir du service météo.



6 Terminer l'interface utilisateur avec une apparence native entre les plateformes

Xamarin.Forms affiche les contrôles de l'interface utilisateur native pour chaque plateforme afin que votre application obtienne automatiquement une apparence native. Pour mieux vous rendre compte, terminez l'interface utilisateur avec un champ d'entrée pour un code postal, puis affichez les données météo retournées par le service.

1. Remplacez le contenu de **WeatherPage.xaml** par le code ci-dessous. Notez que chaque élément est nommé à l'aide de l'attribut **x:Name** comme décrit précédemment

Tutorial : Création d'applications avec Xamarin dans Visual Studio

afin que l'élément puisse être référencé à partir du code. Xamarin.Forms fournit également de nombreuses options de disposition (xamarin.com) ; ici, WeatherPage utilise StackLayout (xamarin.com).

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="WeatherApp.WeatherPage">

  <ContentPage.Resources>
    <ResourceDictionary>
      <Style x:Key="labelStyle" TargetType="Label">
        <Setter Property="TextColor" Value="#a8a8a8" />
        <Setter Property="FontSize" Value="Small" />
      </Style>
      <Style x:Key="fieldStyle" TargetType="Label">
        <Setter Property="TextColor">
          <OnPlatform x:TypeArguments="Color" iOS="Black" Android="White"
WinPhone="White" />
        </Setter>
        <Setter Property="FontSize" Value="Medium" />
      </Style>
      <Style x:Key="fieldView" TargetType="ContentView">
        <Setter Property="Padding" Value="10,0,0,0" />
      </Style>
    </ResourceDictionary>
  </ContentPage.Resources>

  <ContentPage.Content>
    <ScrollView>
      <StackLayout>
        <StackLayout Orientation="Horizontal"
HorizontalOptions="FillAndExpand"
  BackgroundColor="#545454">
          <StackLayout Padding="10,10,10,10" HorizontalOptions="Start">
            <Label Text="Search by Zip Code" TextColor="White"
FontAttributes="Bold"
  FontSize="Medium" />
            <Label x:Name="zipCodeLabel" Text="Zip Code"
Style="{StaticResource labelStyle}" />
            <Entry x:Name="zipCodeEntry" />
          </StackLayout>
          <StackLayout Padding="0,0,0,10" VerticalOptions="End">
            <Button x:Name="getWeatherBtn" Text="Get Weather"
WidthRequest="185" BorderWidth="1" >
              <!-- Set iOS colors; use defaults on other platforms -->
              <Button.TextColor>
                <OnPlatform x:TypeArguments="Color" iOS="White"/>
              </Button.TextColor>
              <Button.BorderColor>
                <OnPlatform x:TypeArguments="Color" iOS="White"/>
              </Button.BorderColor>
            </Button>
          </StackLayout>
        </StackLayout>
      </ScrollView>
    </ContentPage.Content>
  </ContentPage>
```

Tutorial : Création d'applications avec Xamarin dans Visual Studio

```

<StackLayout Padding="10,10,10,10" HorizontalOptions="Start">
  <Label Text="Location" Style="{StaticResource labelStyle}" />
  <ContentView Style="{StaticResource fieldView}">
    <Label Text="{Binding Title}" Style="{StaticResource fieldStyle}"
  />
  </ContentView>
  <Label Text="Temperature" Style="{StaticResource labelStyle}" />
  <ContentView Style="{StaticResource fieldView}">
    <Label x:Name="tempLabel" Text="{Binding Temperature}"
      Style="{StaticResource fieldStyle}" />
  </ContentView>
  <Label Text="Wind Speed" Style="{StaticResource labelStyle}" />
  <ContentView Style="{StaticResource fieldView}">
    <Label x:Name="windLabel" Text="{Binding Wind}"
  Style="{StaticResource fieldStyle}" />
  </ContentView>
  <Label Text="Humidity" Style="{StaticResource labelStyle}" />
  <ContentView Style="{StaticResource fieldView}">
    <Label x:Name="humidityLabel" Text="{Binding Humidity}"
      Style="{StaticResource fieldStyle}" />
  </ContentView>
  <Label Text="Visibility" Style="{StaticResource labelStyle}" />
  <ContentView Style="{StaticResource fieldView}">
    <Label x:Name="visibilityLabel" Text="{Binding Visibility}"
      Style="{StaticResource fieldStyle}" />
  </ContentView>
  <Label Text="Time of Sunrise" Style="{StaticResource labelStyle}" />
  <ContentView Style="{StaticResource fieldView}">
    <Label x:Name="sunriseLabel" Text="{Binding Sunrise}"
      Style="{StaticResource fieldStyle}" />
  </ContentView>
  <Label Text="Time of Sunset" Style="{StaticResource labelStyle}" />
  <ContentView Style="{StaticResource fieldView}">
    <Label x:Name="sunsetLabel" Text="{Binding Sunset}"
      Style="{StaticResource fieldStyle}" />
  </ContentView>
</StackLayout>
</StackLayout>
</ScrollView>
</ContentPage.Content>
</ContentPage>

```

Notez l'utilisation de la balise **OnPlatform** dans `Xamarin.Forms.OnPlatform` sélectionne une valeur de propriété qui est propre à la plateforme actuelle sur laquelle l'application est exécutée (voir [External XAML Syntax](http://xamarin.com) (xamarin.com)). Nous l'utilisons ici pour définir une couleur de texte différente pour les champs de données : blanc sur Android et Windows Phone, noir sur iOS. Vous pouvez utiliser **OnPlatform** pour toutes les propriétés et tous les types de données afin d'effectuer des réglages spécifiques à la plateforme n'importe où dans votre XAML. Dans le fichier code-behind, vous pouvez utiliser l'[API Device.OnPlatform](#) dans le même but.

2. Dans `WeatherPage.xaml.cs`, remplacez le gestionnaire d'événements `GetWeatherBtn_Clicked` par le code ci-dessous. Ce code vérifie qu'un code postal

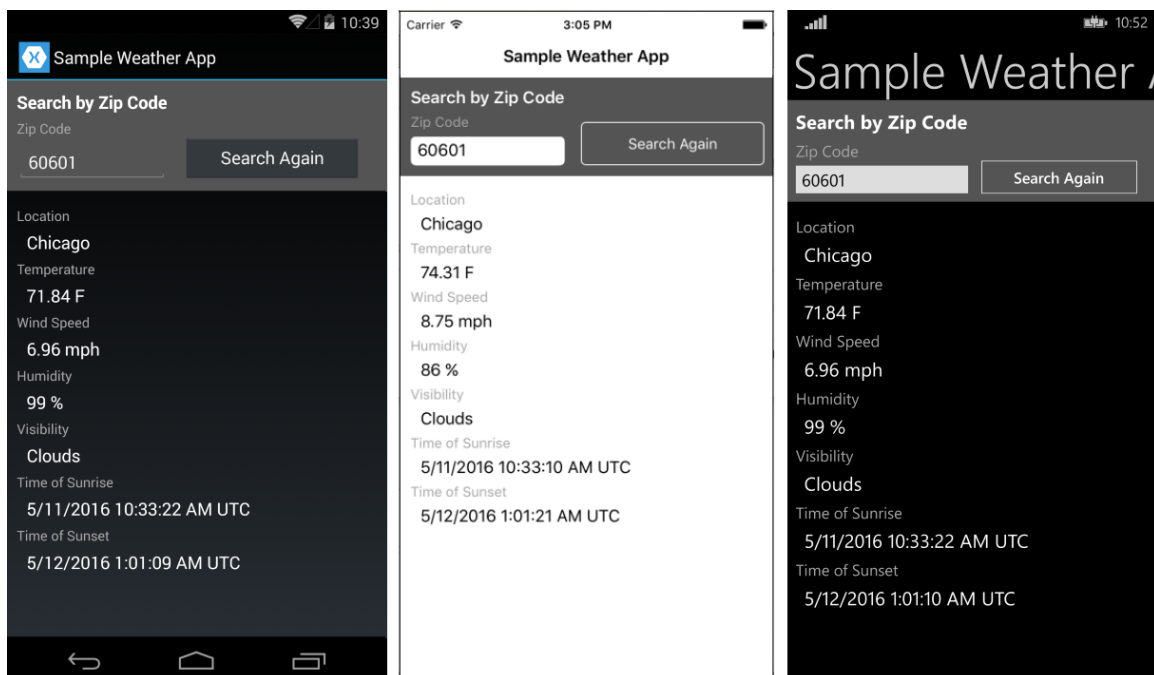
Tutorial : Création d'applications avec Xamarin dans Visual Studio

figure dans le champ d'entrée, récupère les données pour ce code postal, affecte l'instance résultante de **Weather** au contexte de liaison de la totalité de l'écran, puis définit « Rechercher à nouveau » comme texte du bouton. Notez que chaque étiquette dans l'interface utilisateur est liée à une propriété de la classe **Weather** ; par conséquent, quand vous affectez une instance de **Weather** au contexte de liaison de l'écran, ces étiquettes sont automatiquement mises à jour.

```

private async void GetWeatherBtn_Clicked(object sender, EventArgs e)
{
    if (!String.IsNullOrEmpty(zipCodeEntry.Text))
    {
        Weather weather = await Core.GetWeather(zipCodeEntry.Text);
        this.BindingContext = weather;
        getWeatherBtn.Text = "Search Again";
    }
}
  
```

- Exécutez l'application sur les trois plateformes (Android, iOS et Windows Phone) en cliquant avec le bouton droit sur le projet approprié, en sélectionnant Définir comme projet de démarrage et en démarrant l'application sur un appareil, dans l'émulateur ou dans le simulateur. Entrez un code postal des États-Unis valide (par exemple, 60601) et appuyez sur le bouton Get Weather pour afficher les données météo de cette région, comme indiqué ci-dessous. Bien entendu, Visual Studio doit être connecté à un ordinateur Mac OS X sur votre réseau pour le projet iOS.



Tutorial : Création d'applications avec Xamarin dans Visual Studio

7 Mettre en place un Web serveur REST en Xamarin

Question 1: A partir des indications de

<https://github.com/liveowl/Simple-Http-Listener-PCL/blob/master/README.md>, créez votre propre serveur Web sur votre téléphone et testez-le par le renvoi d'une donnée sur un GET (les bases d'un service REST).