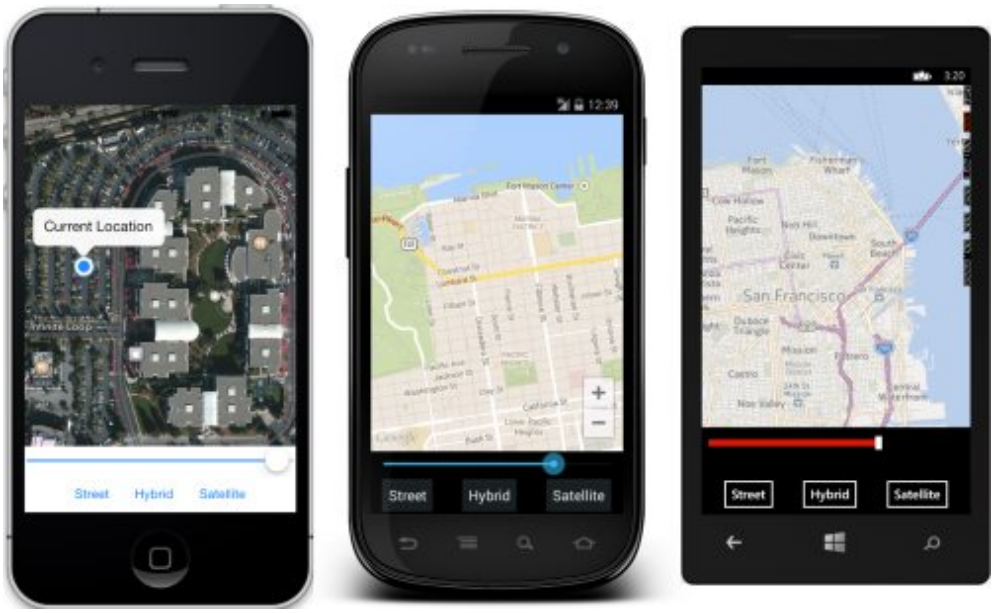
Map Control

Adding a Map in Xamarin.Forms

Xamarin.Forms.Maps uses the native map APIs on each platform. This provides a fast, familiar maps experience for users, but means that some configuration steps are needed to adhere to each platforms specific API requirements. Once configured, the `Map` control works just like any other Xamarin.Forms element in common code.

- Maps Initialization - Using `Map` requires additional initialization code at startup.
- Platform Configuration - Each platform requires some configuration for maps to work.
- Using Maps in C# - Displaying maps and pins using C#.
- Using Maps in XAML - Displaying a map with XAML.

The map control has been used in the MapsSample sample, which is shown below.



Map functionality can be further enhanced by creating a map custom renderer.

Maps Initialization

When adding maps to a Xamarin.Forms application, **Xamarin.Forms.Maps** is a a separate NuGet package that you should add to every project in the solution. On Android, this also has a dependency on GooglePlayServices (another NuGet) which is downloaded automatically when you add Xamarin.Forms.Maps.

After installing the NuGet package, some initialization code is required in each application project, *after* the `Xamarin.Forms.Forms.Init` method call. For iOS use the following code:

```
Xamarin.FormsMaps.Init();
```

On Android you must pass the same parameters as `Forms.Init`:

```
Xamarin.FormsMaps.Init(this, bundle);
```

For the Windows Runtime (WinRT) and the Universal Windows Platform (UWP) use the following code:

```
Xamarin.FormsMaps.Init("INSERT_AUTHENTICATION_TOKEN_HERE");
```

Add this call in the following files for each platform:

- **iOS** - AppDelegate.cs file, in the `FinishedLaunching` method.
- **Android** - MainActivity.cs file, in the `OnCreate` method.
- **WinRT and UWP** - MainPage.xaml.cs file, in the `MainPage` constructor.

Once the NuGet package has been added and the initialization method called inside each applcation, `Xamarin.Forms.Maps` APIs can be used in the common PCL or Shared Project code.

Platform Configuration

Additional configuration steps are required on some platforms before the map will display.

iOS

On iOS 7 the map control "just works", so long as the `FormsMaps.Init()` call has been made.

For iOS 8 two keys need to be added to the **Info.plist** file: `NSLocationAlwaysUsageDescription` and `NSLocationWhenInUseUsageDescription`. The XML representation is shown below - you should update the `string` values to reflect how your application is using the location information:

```
<key>NSLocationAlwaysUsageDescription</key>
    <string>Can we use your location</string>
<key>NSLocationWhenInUseUsageDescription</key>
    <string>We are using your location</string>
```

The plist entries can also be added in **Source** view while editing the **Info.plist** file:

NSLocationWhenInUseUsageDescription	String	We are using your location
NSLocationAlwaysUsageDescription	String	Can we use your location

Android

To use the Google Maps API v2 on Android you must generate an API key and add it to your Android project. Follow the instructions in the Xamarin doc on obtaining a Google Maps API v2 key. After following those instructions, paste the API key in the **Properties/AndroidManifest.xml** file (view source and find/update the following element):

```
<meta-data android:name="com.google.android.maps.v2.API_KEY"
            android:value="AbCdEfGhIjKlMnOpQrStUvWValueGoesHere" />
```

Without a valid API key the maps control will display as a grey box on Android.

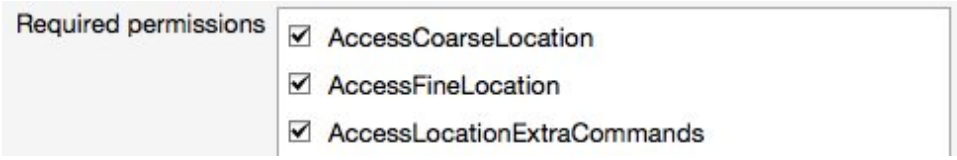
Remember to generate another key using the **keystore** file that is used to sign the Release version of any application that is uploaded to the Google Play store. The key you generate

for development and debugging will not work and the app downloaded from Google Play will have broken map display. Also remember to regenerate the key if the app's Package Name changes.

You'll also need to enable appropriate permissions by right-clicking on the Android project and selecting **Options > Build > Android Application** and ticking the following:

- `AccessCourseLocation`
- `AccessFineLocation`
- `AccessLocationExtraCommands`
- `AccessMockLocation`
- `AccessNetworkState`
- `AccessWifiState`
- `Internet`

Some of these are shown in the screenshot below:



The last two are required because applications require a network connection to download map data. Read about Android permissions to learn more.

Windows Runtime and Universal Windows Platform

To use maps on the Windows Runtime and the Universal Windows Platform you must generate an authorization token. For more information, see Request a maps authentication key on MSDN.

The authentication token should then be specified in the `FormsMaps.Init("AUTHORIZATION_TOKEN")` method call, in order to authenticate the app with Bing Maps.

Using Maps

See the MapPage.cs in the MobileCRM sample for an example of how the map control can be used in code. A simple `MapPage` class might look like this - notice that a new `MapSpan` is created to position the map's view:

```
public class MapPage : ContentPage {
    public MapPage() {
        var map = new Map(
            MapSpan.FromCenterAndRadius(
                    new Position(37,-122), Distance.FromMiles(0.3)))
{
                IsShowingUser = true,
                HeightRequest = 100,
                WidthRequest = 960,
                VerticalOptions = LayoutOptions.FillAndExpand
            };
        var stack = new StackLayout { Spacing = 0 };
        stack.Children.Add(map);
        Content = stack;
    }
}
```

Map Type

The map content can also be changed by setting the `MapType` property, to show a regular street map (the default), satellite imagery or a combination of both.

```
map.MapType == MapType.Street;
```

Valid `MapType` values are:

- Hybrid

- Satellite
- Street (the default)

Map Region and MapSpan

As shown in the code snippet above, supplying a `MapSpan` instance to a map constructor sets the initial view (center point and zoom level) of the map when it is loaded. The `MoveToRegion` method on the map class can then be used to changed the position or zoom level of the map. There are two ways to create a new `MapSpan` instance:

- **MapSpan.FromCenterAndRadius()** - static method to create a span from a `Position` and specifying a `Distance` .
- **new MapSpan ()** - constructor that uses a `Position` and the degress of latitude and longitude to display.

To change the zoom level of the map without altering the location, create a new `MapSpan` using the current location from the `VisibleRegion.Center` property of the map control. A `Slider` could be used to control map zoom like this (however zooming directly in the map control cannot currently update the value of the slider):

```
var slider = new Slider (1, 18, 1);
slider.ValueChanged += (sender, e) => {
    var zoomLevel = e.NewValue; // between 1 and 18
    var latlongdegrees = 360 / (Math.Pow(2, zoomLevel));
    map.MoveToRegion(new MapSpan (map.VisibleRegion.Center,
latlongdegrees, latlongdegrees));
};
```

Map Pins

Locations can be marked on the map with `Pin` objects.

```
var position = new Position(37,-122); // Latitude, Longitude
var pin = new Pin {
            Type = PinType.Place,
            Position = position,
            Label = "custom pin",
            Address = "custom detail info"
        };
map.Pins.Add(pin);
```

`PinType` can be set to one of the following values, which may affect the way the pin in rendered (depending on the platform):

- Generic
- Place
- SavedPin
- SearchResult

Using Xaml

Maps can also be positioned in Xaml layouts as shown in this snippet.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
```

```
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:maps="clr-
namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"
    x:Class="MapDemo.MapPage">
    <StackLayout VerticalOptions="StartAndExpand" Padding="30">
        <maps:Map WidthRequest="320" HeightRequest="200"
            x:Name="MyMap"
            IsShowingUser="true"
            MapType="Hybrid"
        />
    </StackLayout>
</ContentPage>
```

The `MapRegion` and `Pins` can be set in code using the `MyMap` reference (or whatever the map is named). Note that an additional `xmlns` namespace definition is required to reference the Xamarin.Forms.Maps controls.

```
MyMap.MoveToRegion(
    MapSpan.FromCenterAndRadius(
        new Position(37,-122), Distance.FromMiles(1)));
```

Summary

The Xamarin.Forms.Maps is a separate NuGet that must be added to each project in a Xamarin.Forms solution. Additional initialization code is required, as well as some configuration steps for iOS, Android, WinRT, and UWP.

Once configured the Maps API can be used to render maps with pin markers in just a few lines of code. Maps can be further enhanced with a custom renderer.