

Universal Windows Apps

# Overview

In this demo, we will introduce the Universal Windows Platform (UWP) on Windows 10. Now you can build one app that runs on all Windows 10 devices and design your pages so they render properly no matter what device is used to view them. Run your app on a Windows 10 phone, a Windows 10 desktop, or Xbox. With the introduction of the Windows 10 core and the Universal Windows Platform (UWP), one app package can run across all platforms.

# What You’ll Learn

* You will build a simple map application that runs on all Windows 10 devices.

# Tools You’ll Use

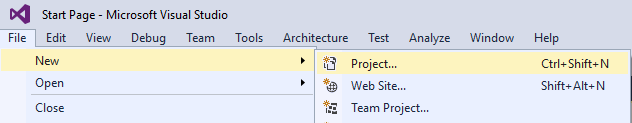
* Visual Studio 2015. You can use any of the following versions:
  + Visual Studio Express 2015 for Windows
  + Visual Studio Community 2015
  + Visual Studio Professional 2015
  + Visual Studio Enterprise 2015

# The Challenge

We will start by creating a new project in Visual Studio. Launch Visual Studio 2015 from the task bar or the Start Menu.

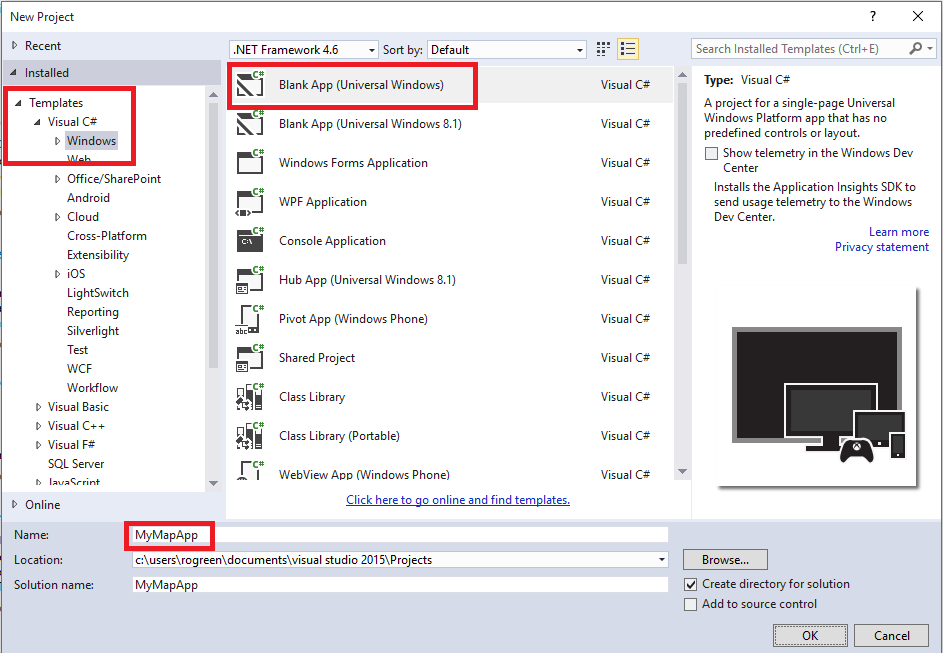


From the *File Menu*, select *New Project*.

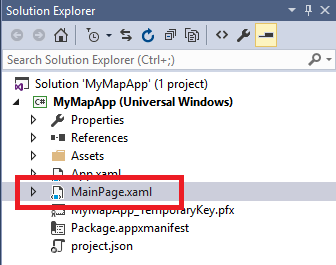


Using the New Project window, you will create a blank Universal Windows App from the provided project templates.

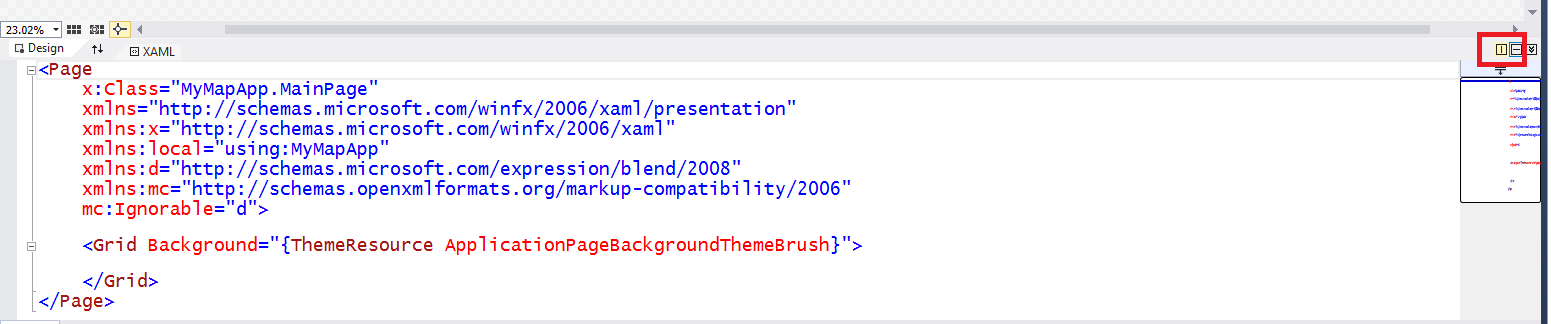
Expand the template tree on the left-hand side of the menu to get to the *Windows* templates for *Visual C#*: *Installed 🡪 Templates 🡪 Visual C# 🡪 Windows.* Select the *Blank App (Universal Windows)* template.Name the project *MyMapApp* and click *OK*.

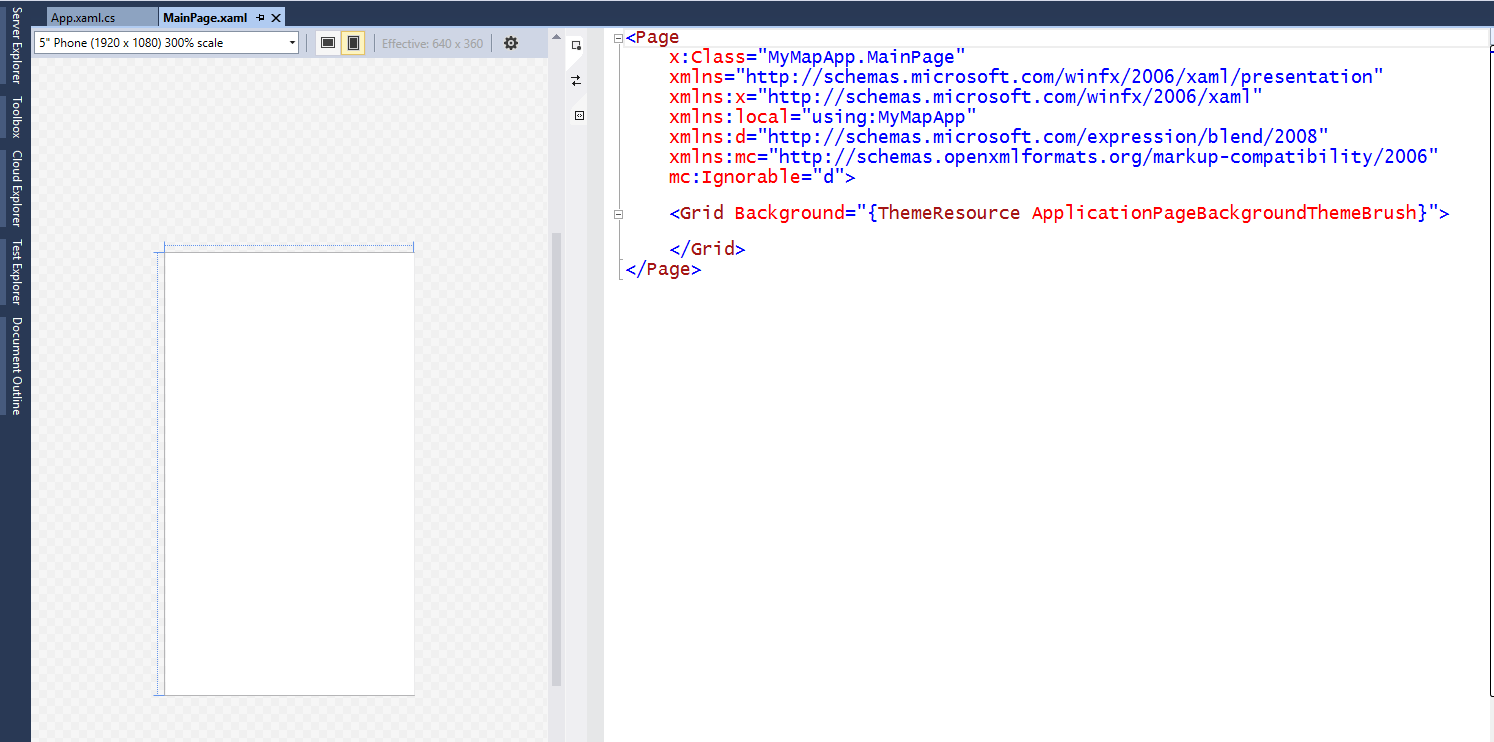


In the *Solution Explorer*, double-click the *MainPage.xaml* file.

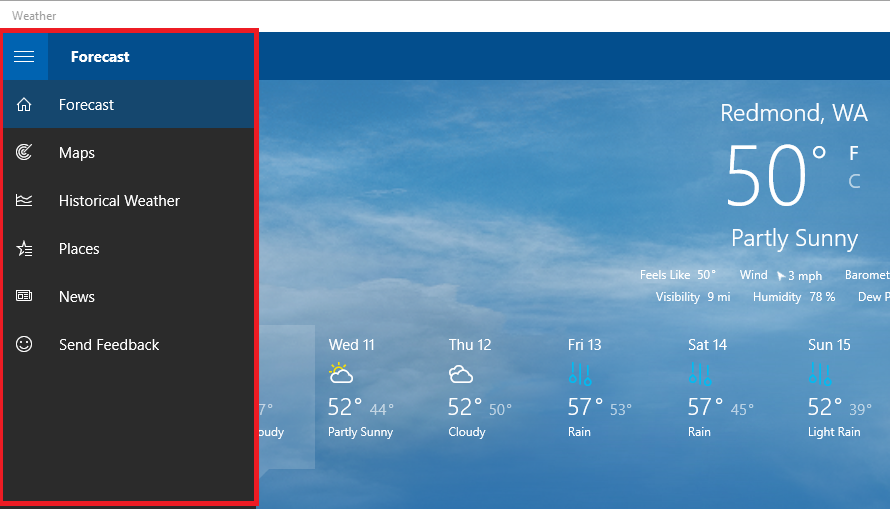


Click the Vertical Split button to see both the XAML and the screen side by side.





To build the Map app, we’ll start first with creating a SplitView control, often referred to as a “Hamburger control”. This SplitView is a navigation control that can adapt its design to optimize screen real estate for content. Here is how the Weather app uses a SplitView.



Add a SplitView control to the page. Put the following XAML between the opening and closing Grid elements.

<SplitView x:Name="MySplitView"

DisplayMode="Inline"

IsPaneOpen="True">

</SplitView>

The SplitView’s pane contains the navigation items. The DisplayMode property specifies how the pane is displayed. We are setting this to Inline so that the pane side-displays by-side with the rest of the content on the page. We are also specifying that we want the pane to be open by default.

Next we need to add navigation items. Add the following XAML between the opening and closing SplitView elements.

<SplitView.Pane>

<StackPanel Margin="0,30,0,0">

<RadioButton Tag="Map"

Click="RadioButtonPaneItem\_Click">

<StackPanel Orientation="Horizontal">

<SymbolIcon Symbol="Map" />

<TextBlock Text="Map"

Margin="15,0,0,0" />

</StackPanel>

</RadioButton>

<RadioButton Tag="Mail"

Click="RadioButtonPaneItem\_Click">

<StackPanel Orientation="Horizontal">

<SymbolIcon Symbol="Mail" />

<TextBlock Text="Mail"

Margin="15,0,0,0" />

</StackPanel>

</RadioButton>

<RadioButton Tag="AddFriend"

Click="RadioButtonPaneItem\_Click">

<StackPanel Orientation="Horizontal">

<SymbolIcon Symbol="AddFriend" />

<TextBlock Text="Add a Friend"

Margin="15,0,0,0" />

</StackPanel>

</RadioButton>

<RadioButton Tag="Settings"

Click="RadioButtonPaneItem\_Click">

<StackPanel Orientation="Horizontal">

<SymbolIcon Symbol="Setting" />

<TextBlock Text="Settings"

Margin="15,0,0,0" />

</StackPanel>

</RadioButton>

</StackPanel>

</SplitView.Pane>

The SplitView, by default does not come with a Hamburger icon, so let's create one. Add the following XAML below the SplitView.

<Button x:Name="HamburgerButton"

Width="{Binding ElementName=MySplitView, Path=CompactPaneLength}"

VerticalAlignment="Top"

Click="HamburgerButton\_Click">

<StackPanel>

<Rectangle Width="15"

Height="2"

Margin="2"

Fill="White" />

<Rectangle Width="15"

Height="2"

Margin="2"

Fill="White" />

<Rectangle Width="15"

Height="2"

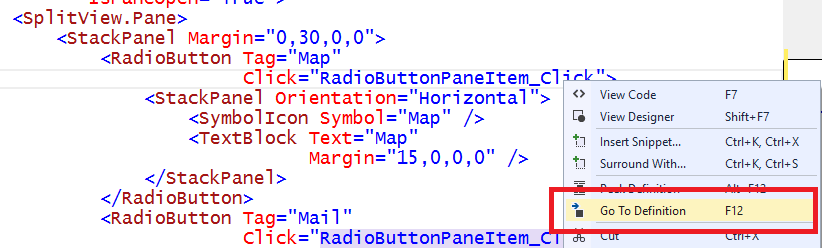
Margin="2"

Fill="White" />

</StackPanel>

</Button>

Right-click on RadioButtonPaneItem\_Click and select *Go To Definition*.



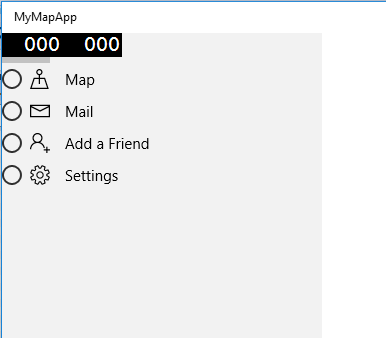
We will add code to this method later.

Return to MainPage.xaml. Right-click on HamburgerButton\_Click and select *Go To Definition*.

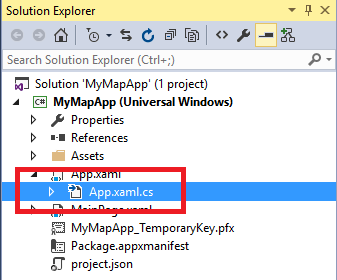
Add the following code to the HamburgerButton\_Click method to open and close the SplitView pane when you click the Hamburger control.

MySplitView.IsPaneOpen = MySplitView.IsPaneOpen ? false : true;

Press *F5* to build and run the app. The SplitView pane is open and displays the navigation items. The hamburger control is hidden by the black rectangle with numbers. Close the app and return to Visual Studio.



To get rid of the black box with numbers, open App.xaml.cs. If it is not still open, double-click on in the Solution Explorer.



Locate the OnLaunched method and comment out the code that enables the frame rate counter.

if (System.Diagnostics.Debugger.IsAttached)

{

//this.DebugSettings.EnableFrameRateCounter = true;

}

In the Solution Explorer, double-click App.xaml. To change the default style of the radio buttons in the navigation menu, add the following XAML.

<Application.Resources>

<ResourceDictionary>

<Style TargetType="RadioButton">

<Setter Property="Template">

<Setter.Value>

<ControlTemplate>

<ContentPresenter Margin="15" />

</ControlTemplate>

</Setter.Value>

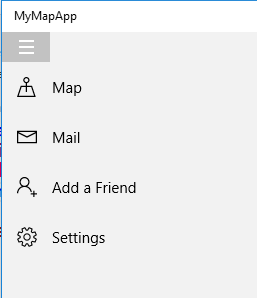
</Setter>

</Style>

</ResourceDictionary>

</Application.Resources>

Press *F5* to run the app. You can now see the Hamburger control. Notice also that the circles next to the navigation items are no longer visible. Click the Hamburger control and the pane closes.



Close the app and return to Visual Studio.

Return to MainPage.xaml.

You will need a place for the app’s content, so use the highlighted XAML below to add a Frame element to the SplitView.

</SplitView.Pane>

<SplitView.Content>

<Frame x:Name="MainFrame" />

</SplitView.Content>

</SplitView>

Right-click in the XAML and select *View Code*.

To wire up the navigation, add the following code to the RadioButtonPaneItem\_Click method.

var radioButton = sender as RadioButton;

if (radioButton != null)

{

switch (radioButton.Tag.ToString())

{

case "Map":

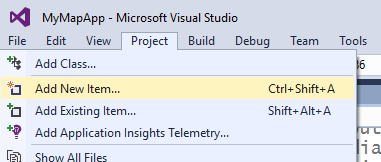
MainFrame.Navigate(typeof(MapPage));

break;

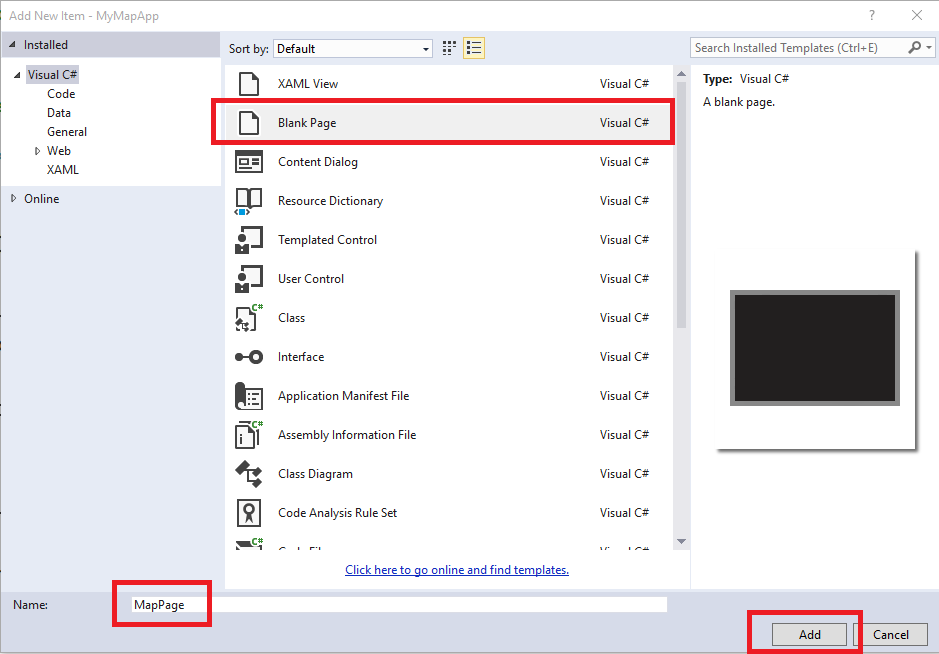
}

}

You will now create the map page. From the *Project* menu, select *Add New Item*.



In the Add New Item dialog, select the *Blank Page* template. Set the Name to *MapPage* and click *Add*.



Replace the XAML with the following.

<Page x:Class="MyMapApp.MapPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="using:MyMapApp"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:maps="using:Windows.UI.Xaml.Controls.Maps"

mc:Ignorable="d">

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

<StackPanel>

<StackPanel x:Name="SearchControls"

Orientation="Horizontal">

<CheckBox x:Name="TrafficCheckBox"

Content="Show traffic"

Width="100"

Height="50"

Margin="15,35,15,15"

Checked="TrafficCheckBox\_Checked"

Unchecked="TrafficCheckBox\_Unchecked"/>

<Button x:Name="MapStyleButton"

Content="Aerial"

Width="100"

Height="50"

Margin="15"

Click="MapStyleButton\_Click"/>

</StackPanel>

<maps:MapControl x:Name="MapControl"

Height=”500"/>

</StackPanel>

</Grid>

</Page>

Right-click anywhere in the XAML and select *View Code* from the popup menu.

Replace the code with the following:

using System;

using Windows.Devices.Geolocation;

using Windows.UI.Popups;

using Windows.UI.Xaml;

using Windows.UI.Xaml.Controls;

namespace MyMapApp

{

public sealed partial class MapPage : Page

{

public MapPage()

{

this.InitializeComponent();

MapControl.Loaded += MapControl\_Loaded;

MapControl.MapTapped += MapControl\_MapTapped;

}

private void MapControl\_Loaded(object sender, RoutedEventArgs e)

{

MapControl.Center =

new Geopoint(new BasicGeoposition()

{

//Geopoint for Seattle

Latitude = 47.604,

Longitude = -122.329

});

MapControl.ZoomLevel = 12;

}

private void TrafficCheckBox\_Checked(object sender, RoutedEventArgs e)

{

MapControl.TrafficFlowVisible = true;

}

private void TrafficCheckBox\_Unchecked(object sender, RoutedEventArgs e)

{

MapControl.TrafficFlowVisible = false;

}

private void MapStyleButton\_Click(object sender, RoutedEventArgs e)

{

if (MapControl.Style ==

Windows.UI.Xaml.Controls.Maps.MapStyle.Aerial)

{

MapControl.Style =

Windows.UI.Xaml.Controls.Maps.MapStyle.Road;

MapStyleButton.Content = "Aerial";

}

else

{

MapControl.Style =

Windows.UI.Xaml.Controls.Maps.MapStyle.Aerial;

MapStyleButton.Content = "Road";

}

}

private async void MapControl\_MapTapped(

Windows.UI.Xaml.Controls.Maps.MapControl sender,

Windows.UI.Xaml.Controls.Maps.MapInputEventArgs args)

{

var tappedGeoPosition = args.Location.Position;

string status =

$"Map tapped at \nLatitude: {tappedGeoPosition.Latitude} " +

$"\nLongitude: {tappedGeoPosition.Longitude}";

var messageDialog = new MessageDialog(status);

await messageDialog.ShowAsync();

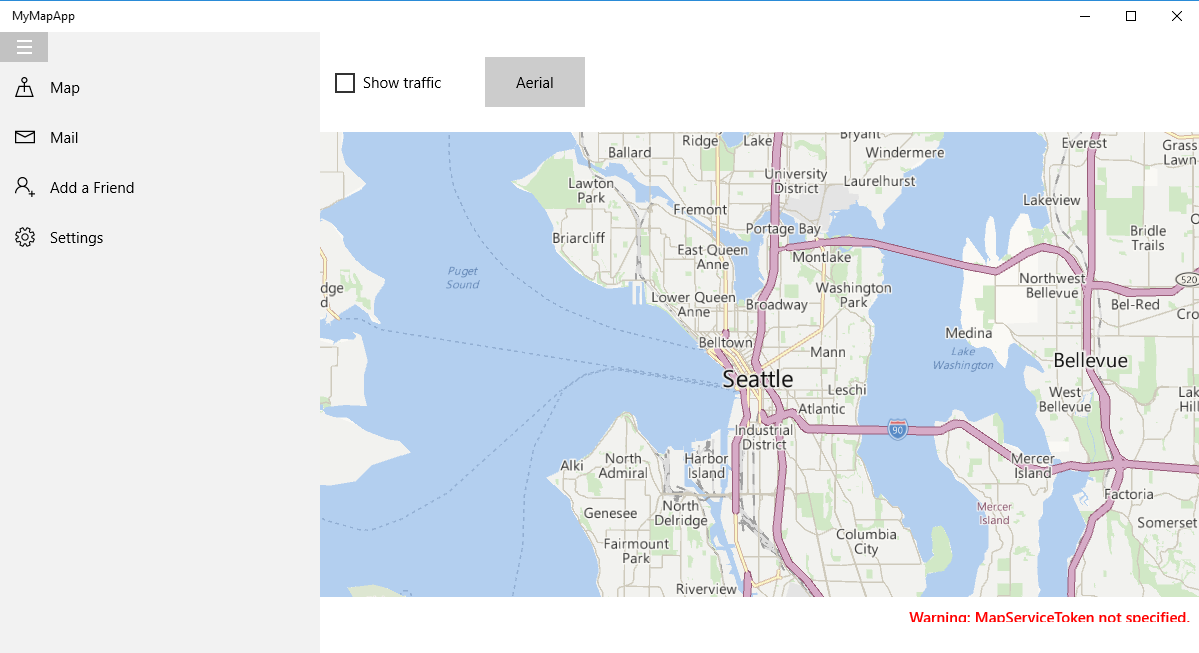
}

}

}

Press *F5* to run the app.

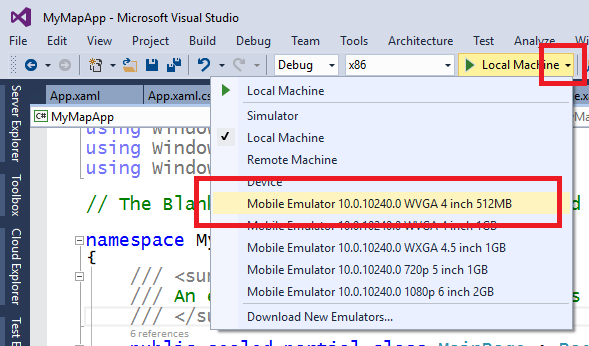
Click *Map* to view the map.



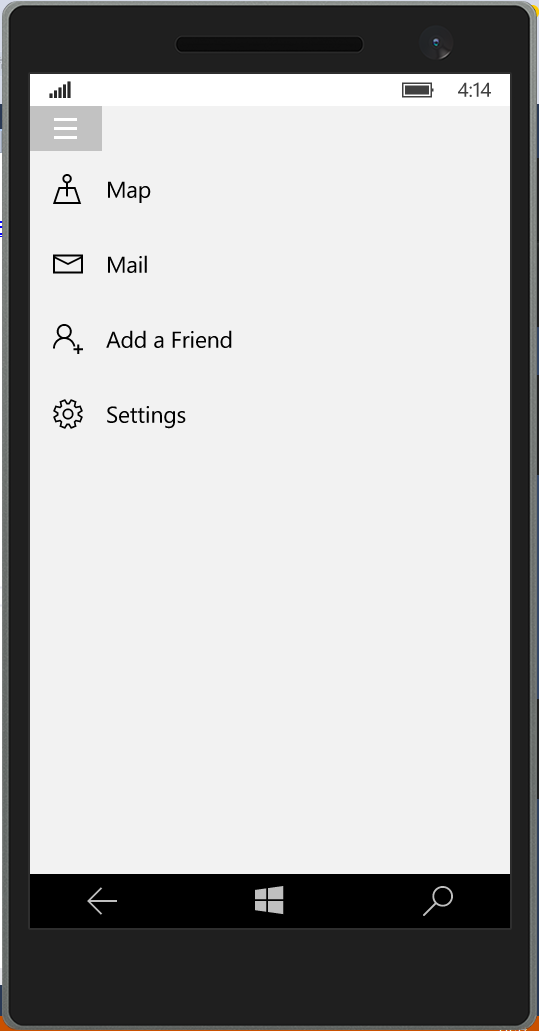
Check and uncheck *Show traffic* to see or hide Seattle traffic. Click *Aerial* to see an aerial view of Seattle. Click *Road* to return to the roads view.

Close the app and return to Visual Studio.

Let’s see how this app looks when it runs on a phone. Click the arrow next to Local Machine to display the list of app targets. Select any of the emulators.

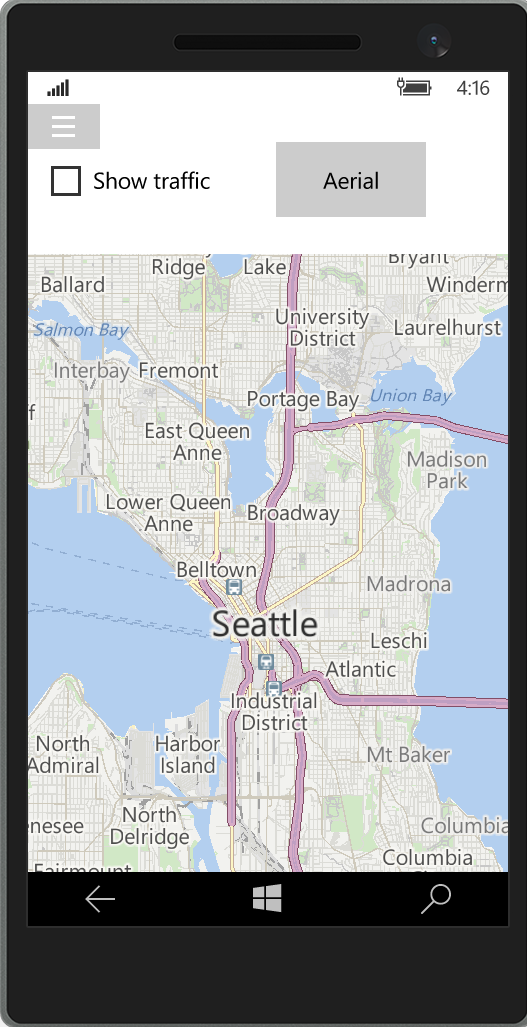


Press *F5* to run the app. Once the Windows Mobile emulator starts you should see the app.



Clicking the Hamburger control will open and close the pane.

Click *Map* to view the map. Click the Hamburger control to close the pane, which is hiding the map page.



Return to Visual Studio and press *Shift + F5* to stop the app. Leave the emulator open.

Return to MainPage.xaml.cs.

Make the following change to the RadioButtonPaneItem\_Click method to hide the navigation pane after making a selection.

if (radioButton != null)

{

switch (radioButton.Tag.ToString())

{

case "Map":

MainFrame.Navigate(typeof(MapPage));

break;

}

MySplitView.IsPaneOpen = false;

}

**Visual State Manager and Adaptive Triggers**

When you create a Universal Windows app, it will run on any device that runs Windows 10. You only have to create the pages in the app once, rather than once for each device. However, for the best user experience, you want the pages to look different depending on where the app is running. This is known as responsive design or adaptive UI. The good news is that you can accomplish this by adding some XAML to each page.

First, let’s have the SplitView behave differently based on the width of the screen. Open MainPage.xaml.

Add the following XAML between the opening Grid and SplitView elements.

<VisualStateManager.VisualStateGroups>

<VisualStateGroup>

<VisualState x:Name="Narrow">

<VisualState.StateTriggers>

<AdaptiveTrigger MinWindowWidth="0" />

</VisualState.StateTriggers>

<VisualState.Setters>

<Setter Target="MySplitView.DisplayMode"

Value="Overlay" />

<Setter Target="MySplitView.IsPaneOpen"

Value="False" />

</VisualState.Setters>

</VisualState>

<VisualState x:Name="Compact">

<VisualState.StateTriggers>

<AdaptiveTrigger MinWindowWidth="720" />

</VisualState.StateTriggers>

<VisualState.Setters>

<Setter Target="MySplitView.DisplayMode"

Value="CompactOverlay" />

<Setter Target="MySplitView.IsPaneOpen"

Value="False" />

</VisualState.Setters>

</VisualState>

<VisualState x:Name="Wide">

<VisualState.StateTriggers>

<AdaptiveTrigger MinWindowWidth="1024"/>

</VisualState.StateTriggers>

<VisualState.Setters>

<Setter Target="MySplitView.DisplayMode"

Value="CompactInline" />

<Setter Target="MySplitView.IsPaneOpen"

Value="True" />

</VisualState.Setters>

</VisualState>

</VisualStateGroup>

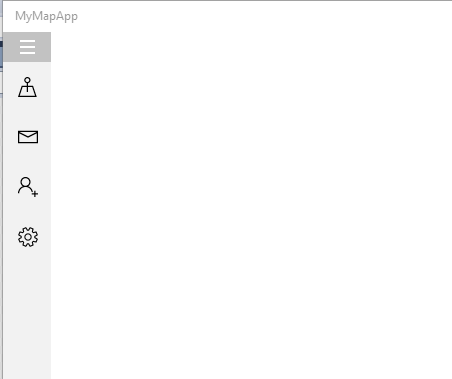
</VisualStateManager.VisualStateGroups>

The XAML changes the DisplayMode and IsPaneOpen properties of the SplitView at runtime based on size of the app’s window.

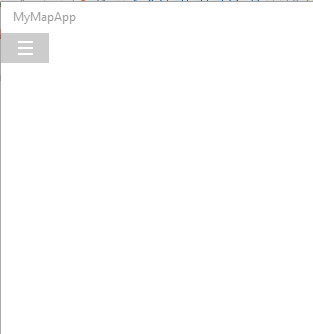
Click the arrow next to Mobile Emulator to display the list of app targets. Select *Local Machine*.

Press *F5* to run the app.

Make the window narrower. When it is 1024 pixels the navigation menu changes to display only icons.



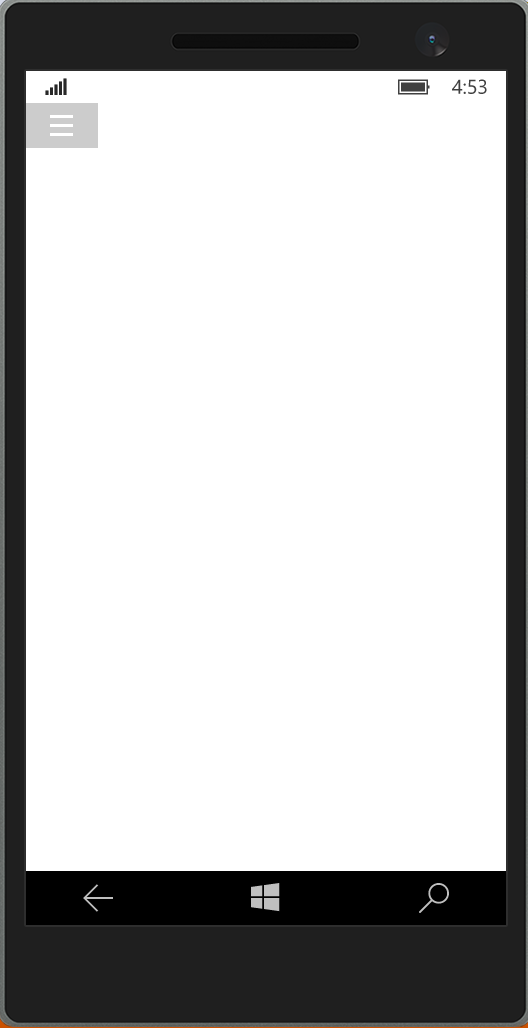
Make the window even narrower. When it is 720 pixels the SplitView pane is closed and the menu disappears.



If you click the Hamburger control however, the pane opens and the navigation menu displays both icons and text.

Close the app and return to Visual Studio.

Click the arrow next to Local Machine to display the list of app targets. Select the same emulator you selected before. The screen width on a phone is quite narrow, so the SplitView pane is closed when the app starts. Click the Hamburger control to open it.



Return to Visual Studio and press *Shift + F5* to stop the app. Leave the emulator open.

The last thing we will do to the app is add XAML to the Map page to have it display differently on different devices.

Open MapPage.xaml.

Change the first StackPanel element to a RelativePanel.

<RelativePanel>

<StackPanel x:Name="SearchControls"

Orientation="Horizontal">

...

<maps:MapControl x:Name="MapControl"

Height="500" />

</RelativePanel>

Next modify the map control so that it automatically aligns itself with the left and bottom of the relative panel. It will also take up the remaining space.

<maps:MapControl x:Name="MapControl"

RelativePanel.AlignBottomWithPanel="True"

RelativePanel.AlignLeftWithPanel="True"/>

Add the following XAML between the opening Grid and RelativePanel elements.

<VisualStateManager.VisualStateGroups>

<VisualStateGroup>

<VisualState x:Name="Narrow">

<VisualState.StateTriggers>

<AdaptiveTrigger MinWindowWidth="0" />

</VisualState.StateTriggers>

<VisualState.Setters>

<Setter Target="SearchControls.(RelativePanel.AlignRightWithPanel)"

Value="True" />

<Setter Target="MapControl.(RelativePanel.Below)"

Value="SearchControls" />

<Setter Target="MapControl.(RelativePanel.AlignRightWithPanel)"

Value="True" />

</VisualState.Setters>

</VisualState>

<VisualState x:Name="Wide">

<VisualState.StateTriggers>

<AdaptiveTrigger MinWindowWidth="720" />

</VisualState.StateTriggers>

<VisualState.Setters>

<Setter Target="SearchControls.(RelativePanel.AlignRightWithPanel)"

Value="True" />

<Setter Target="MapControl.(RelativePanel.AlignTopWithPanel)"

Value="True" />

<Setter Target="MapControl.(RelativePanel.LeftOf)"

Value="SearchControls" />

</VisualState.Setters>

</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

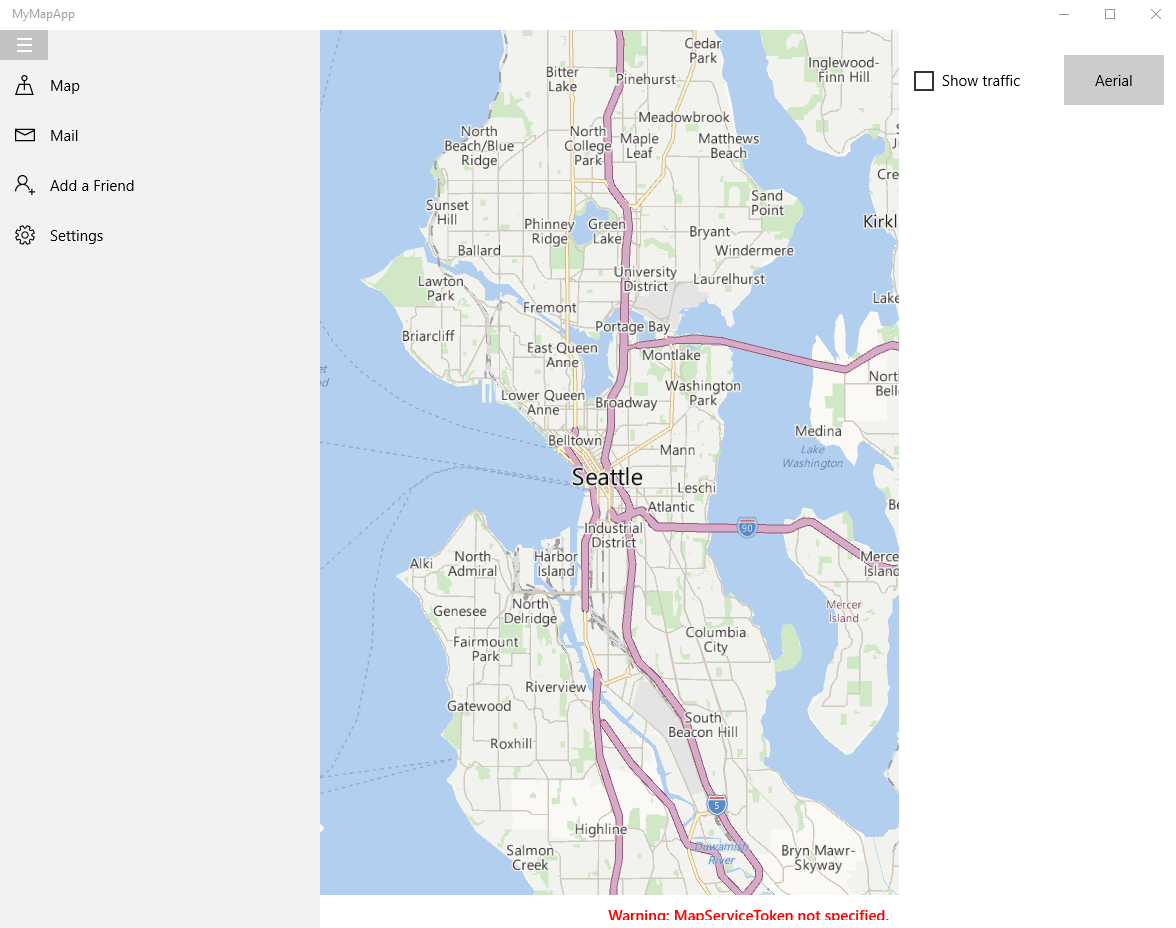
Click the arrow next to Mobile Emulator to display the list of app targets. Select *Local Machine*.

Press *F5* to run the app.

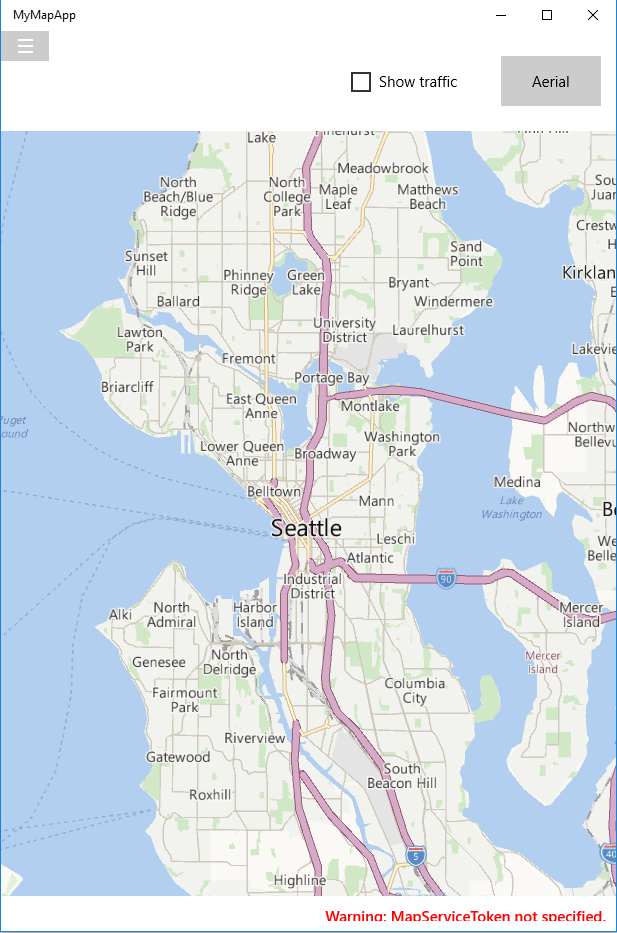
Resize the window so that the StackPanel pane displays both icons and text.

Click *Map*.

The map is sized to the height of the windows and the controls display to the left.



Make the window even narrower. When it is 720 pixels the map controls are displayed above the map.

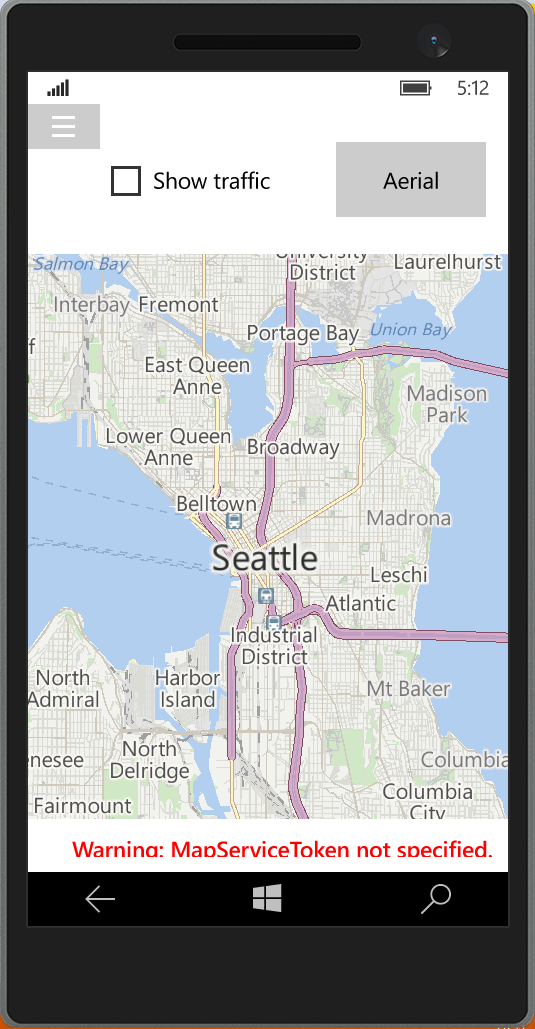


Close the app and return to Visual Studio.

Click the arrow next to Local Machine to display the list of app targets. Select the same emulator you selected before.

Press *F5* to run the app.

Click the Hamburger control and then click *Map*. Click the Hamburger control to



## Wrap Up

Congratulations on completing this challenge.

# Want more?

Visit [www.microsoftvirtualacademy.com](http://www.microsoftvirtualacademy.com) and check out our full collection of Quick Start Challenges.