

Location Intelligence for Windows Store Apps

A complete guide to creating the future of Location Aware apps



Ricky Brundritt

Location Intelligence for Windows Store Apps

Copyright © 2014 by Ricky Brundritt

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review or scholarly journal.

ISBN: 978-1-291-76109-2

<http://rbrundritt.wordpress.com>

I dedicate this book to my wife, Rose, who has always been there for me with patience, love, and friendship.

- Ricky Brundritt

Contents at a Glance

About the Author.....	i
Acknowledgements	ii
Introduction	iii
Chapter 1: Getting Started	1
Chapter 2: The Sensor and Location Platform	27
Chapter 3: Bing Maps JavaScript API.....	49
Chapter 4: Bing Maps Native API.....	103
Chapter 5: Bing Maps REST Services	154
Chapter 6: Bing Spatial Data Services	195
Chapter 7: Working with Spatial Data.....	235
Chapter 8: Drawing on the Map	276
Chapter 9: Creating an Augmented Reality App	304
Chapter 10: Creating a Templatable Compass Control	333
Chapter 11: Cross Platform Development.....	344
Appendix A: ISO-3166 Country Codes.....	385
Appendix B: Bing Maps REST Services Class Diagrams	388
Appendix C: Bing Maps REST Service Pushpin Icon Styles.....	394
Appendix D: GeoData API Data Contracts	397
Glossary.....	401

Table of Contents

About the Author.....	i
Acknowledgements	ii
Introduction	iii
Who this book is for?	iii
Chapter Overview.....	iii
Prerequisites.....	iv
Design Tip	v
Downloading the code	v
Contacting the author.....	vi
Recommended Reading.....	vi
Chapter 1: Getting Started	1
Introduction to Windows Store Apps.....	1
Getting a developer license	2
Creating a Windows Store App	2
Updating the app manifest.....	5
Using the Windows Store App Simulator	6
Side Loading a Windows Store app	8
Linking to the Maps app	9
Creating a Uri to the Maps App.....	10
Launching the Maps App.....	12
Bing Maps Platform.....	16
Key Features	17
Free Terms of Use	18
Creating a Bing Maps account.....	18
Creating a Bing Maps key.....	19
The Bing Maps Tile System	19
Design Tip: Segoe UI Symbol Buttons	21
Real World Example: The British Airways Inspiration App	24
Chapter Summary	24
Chapter 2: The Sensor and Location Platform.....	27
Overview of the Sensor API	27
Accelerometer.....	31
Compass.....	31
Gyrometer.....	32

Inclinometer.....	32
Light Sensor	32
OrientationSensor	33
SimpleOrientationSensor	34
The Location Platform	34
Alternate Location Providers	38
Geofencing API.....	39
Testing Geofences.....	47
Real World Example: Jack of Tools	47
Chapter Summary	48
Chapter 3: Bing Maps JavaScript API.....	49
Bing Maps Modular Framework	49
Implementing Modules	49
Creating custom modules.....	50
Adding Bing Maps to a Windows Store app.....	52
Customizing the Map	59
Customizing the Map when Loading.....	62
The Breadcrumb Control	62
Culture and Localization.....	62
Map Function Methods	63
Changing the Map View.....	64
Overlaying Content on the Map.....	65
Locations, LocationRects and Colors.....	65
Layering Content	67
Pushpins	67
Polylines	71
Polygons.....	72
Advance Shapes.....	74
Infoboxes	75
Tile Layers	78
Traffic Manager.....	81
Search Manager.....	82
Directions Manager.....	87
Venue Maps.....	92
Working with Map Events	94
Displaying a User's Location.....	97

Creating a Custom Module.....	98
Chapter Summary	101
Chapter 4: Bing Maps Native API.....	102
Adding Bing Maps to a Windows Store app.....	103
Customizing the Map	109
Customizing the Map when Loading.....	111
Building Footprints and a Breadcrumb Control	113
Culture and Localization.....	114
Changing the Map View.....	114
Overlaying Content on the Map.....	115
Layering Content	115
Pushpins.....	118
Polylines	119
Polygons.....	121
Tile Layers.....	122
Traffic Information	126
Overlaying Custom User Controls.....	128
Creating Infoboxes.....	130
Search Manager.....	132
Directions Manager.....	135
Venue Maps.....	139
Working with Map Events	141
Creating Draggable Pushpins	143
Chapter Summary	152
Chapter 5: Bing Maps REST Services	154
Bing Maps REST Services Overview.....	154
Locations API.....	155
Geocoding Addresses.....	156
Geocoding a Single Line Address.....	157
Geocoding Tips.....	158
Reverse Geocoding Coordinates.....	158
Reverse Geocoding Tip	159
Routes API	159
Understanding Waypoints.....	160
Calculating Driving & Walking Directions.....	160
Routing Tips.....	161

Imagery API.....	161
Generating Map Images	163
Drawing Maps with Pushpins	164
Drawing Maps with Routes	166
Making Use of Sessions.....	167
Implementing Session Keys	168
Design Tips to Optimize Sessions	168
Sharing Maps using the Share Charm.....	169
Creating a URL to Bing Maps	169
Sharing from a Windows Store App.....	170
Implementing the Bing Maps REST Services.....	178
Implement using JavaScript.....	178
Implement using .NET	179
Making HTTP Get Requests.....	179
Geocoding with the Search Charm.....	180
Creating a basic mapping app.....	180
Integrating with the Search Charm.....	183
Adding the Geocoding logic	187
Real World Example: Meet me in the Middle	192
Chapter Summary	193
Chapter 6: Bing Spatial Data Services	195
The Bing Maps Portal	195
API Usage Limits.....	195
Bing Spatial Data Services Excel Add-in	196
Geocode DataFlow API	196
Data Source Management API	197
Data Source Schema	197
Geometry Data Type.....	199
Public Data Sources	199
Query API.....	199
Find Nearby Query.....	201
Find in Bounding Box Query.....	201
Find Along a Route Query	201
Geometry Intersection Query.....	202
Implementing in JavaScript.....	203
Implement in .NET.....	203

Creating a Point of Interest Search app	205
GeoData API	217
GeoData Response Class	218
Decompressing a GeoData Shape	221
Creating a Boundary search app	224
Chapter Summary	233
Chapter 7: Working with Spatial Data	235
Common Spatial Data Formats	235
Well Known Text	235
Well Known Binary	236
ESRI Shapefiles	236
KML Files	237
GeoRSS Files	237
GPX Files	237
GeoJSON Format	238
Importing Data using JavaScript Modules	238
Implementing the Well Known Text Module	239
Introduction to the Microsoft Maps Spatial Toolbox Library	241
Clustering Tools	244
Heat Map Layer	247
Creating a Spatial Data Viewer in .NET	251
Windows Runtime Components	261
Creating a Spatial Data Viewer in JavaScript	261
Real World Example: ESRI ArcGIS Runtime SDK for .NET (Beta)	267
Connecting Spatial Data to your app	267
Bing Spatial Data Services	268
OneDrive	268
Storing data locally with SQLite	269
Static Tile Layers	269
Dynamic Tile Layers	270
Custom Web Services	271
Windows Azure Mobile Service	272
Real World Example: FloodAlerts	272
Chapter Summary	273
Chapter 8: Drawing on the Map	276
Creating the base project	277

Adding the Map and Paint Control Panel.....	278
Creating a Paint Manager.....	285
Bringing Everything Together.....	297
Taking the application further.....	303
Chapter 9: Creating an Augmented Reality App	304
Sensors in Augmented Reality Apps.....	304
Creating an Augmented Reality App.....	305
Creating the Base Project.....	305
Creating the Application UI.....	307
Initializing the App.....	309
Adding Helper Methods.....	312
Rendering Items in the Augmented Reality View.....	318
Handling App Navigation.....	321
Adding the Sensor Event Handlers.....	323
Testing the App.....	330
Real World Example: SkyMap	332
Chapter 10: Creating a Templatable Compass Control	333
Creating the Compass Control.....	333
Creating the Compass Templates.....	338
Applying the Templates to the Compass Control.....	341
Chapter Summary	343
Chapter 11: Cross Platform Development.....	344
Creating a Cross Platform app using JavaScript.....	344
Creating a Mobile Web App.....	346
Integrating into a Windows Phone App.....	359
Integrating into a Windows Store App.....	362
jQuery Mobile in Apps.....	366
Wrap you apps with PhoneGap.....	366
Creating a Cross Platform app using .NET	367
Creating the base Project.....	367
Defining Conditional Compilation Symbols.....	370
Creating the User Interface	371
Adding the application logic.....	373
Enabling the Capabilities of the apps.....	379
Portable Class Libraries in .NET	382
Chapter Summary	383

Appendix A: ISO-3166 Country Codes.....	385
Appendix B: Bing Maps REST Services Class Diagrams	388
Base Response Class Diagram.....	388
Common Base Classes	388
Resource Class Diagram.....	390
Routing Related Classes.....	391
Imagery Related Classes.....	393
Appendix C: Bing Maps REST Service Pushpin Icon Styles.....	394
Appendix D: GeoData API Data Contracts	397
C#.....	397
Visual Basic	399
Glossary.....	401

About the Author



Ricky Brundritt has been developing with Bing Maps since 2007. Having started out as a consultant for the Bing Maps technical support team, it was here that he began blogging about Bing Maps development as a way to provide answers to commonly asked questions. He quickly moved into a consulting role and spent many years working with some of the largest companies in the world to help them solve their mapping needs. In time his blog became the most widely used technical resource on Bing Maps outside of Microsoft. Being very active on the Bing Maps forums helping developers he eventually became the moderator. Ricky has been a frequent speaker at a number of industry and Microsoft developer events. As a result of his contributions to the Bing Maps development community Ricky was awarded as a Microsoft MVP in 2009 and maintained this status until he took on a role at Microsoft. He co-founded the Bing Maps UK User Group in 2010. Later that year he won a Bing Maps developer contest placing 1st and 3rd having submitted multiple entries and earning himself the title “King of Bing”. Ricky joined the Microsoft team in 2011 as the Bing Map Technical Solution Professional covering the EMEA region. He is a regular contributor to the official Bing Maps blog and is also the Microsoft MVP Lead for Bing Maps.

Twitter: [@rbrundritt](https://twitter.com/rbrundritt)

LinkedIn: <http://uk.linkedin.com/pub/ricky-brundritt/10/951/a04>

Blog: <http://rbrundritt.wordpress.com>

Acknowledgements

Writing a book on Bing Maps has been a goal of mine for some time. However every time I decided to write one I ran into a common issue; the Bing Maps platform was advancing faster than I could write. With Bing Maps being a significant part of Windows 8 this opened up an opportunity to work with a version of Bing Maps that will continue to grow over time while staying fairly backwards compatible with the current version of the Bing Maps SDK I've based this book on. Having worked with Bing Maps for many years, putting together a list of topics I wanted to include in this book was easy, compiling all of the content, figuring out how best to present it, and editing it was a bit more difficult. It was through the help of a number of individuals that I managed to get this book completed. I would like to thank the following individuals who assisted with the chapter reviews, technical reviews, code samples, inspiration, and much more:

Alastair Aitchison
Brian Norman
Britt Johnson
Zach Johnston
Clayton Garlough
Duncan Lawler

Geoffrey Innis
Joe Schwartz
Johannes Kebeck
John Obrien
Kristoffer Henriksson
Morten Neilson

Nicolas Boonaert
Peter Brack
Stuart Taylor
Tony Young

Introduction

Location Intelligence has been one of the fastest growing industries in recent years and continues to grow at an exponential rate. Seventy to eighty percent of all business data has some sort of geospatial context. Many companies want to make use of this data however most of them do not know where to start. Many of these same companies are planning to create applications targeting Windows 8. You may well be reading this book for this very reason.

With Windows 8 you have the ability to create applications that reach across many platforms such as desktops, laptops and tablet devices. A Windows Store app is a new type of application that runs on Windows 8 devices. Unlike traditional desktop apps, a Windows Store app has a single, chrome-less window that fills the entire screen by default, so there are no distractions. In addition to this, these apps can support different layouts and views to create a fluid experience across different screen sizes and orientations. Several different types of input sources are supported, including touch, pen, mouse, and keyboard input. It's also possible for apps to communicate with each other by sharing content in a standard way. Instead of icons, Windows Store apps use live tiles which can be used to display useful, at-a-glance data to the user, without the need for the user to open up the app. Windows Store apps can be written in several different languages including JavaScript, C#, Visual Basic, C++ and C. Apps are distributed through the Windows Store in Windows 8 and gives you the ability to make your app available to millions of people around the world.

In this book we will dive into the world of location intelligence and the different options for creating location aware applications in Windows 8. The first half of the book focuses on learning what tools are available for creating location aware Windows Store applications. The second half of the book uses a more hands on approach by demonstrating how to develop complete end-to-end location aware solutions.

Who this book is for?

This book is aimed at developers who are being introduced to creating location based Windows Store apps using both Web (HTML, CSS3, JavaScript) and Managed (C#, Visual Basic) programming languages. Previous knowledge on creating location based application is not required or needed and all topics are explained from the ground up. This will include the use of Bing Maps and sensors such as the accelerometer, compass, gyro, and location services. It will be assumed that you have a working knowledge of one of these programming languages; JavaScript, C# or Visual Basic. Experience creating Windows Store applications will help but is not required.

Chapter Overview

In the first half of this book each chapter builds on top of the previous such that by the time you reach the end chapter 7 you will have gained a good working knowledge on how to use all the tools available for creating location aware Windows Store app. The chapters in the second half of the book are independent of each other. Each of these chapters show how to create a complete end to end application. If you wish to skip between these chapters, you can do so without missing out on any content that might be required by a later chapter. Here is a brief overview of what will be covered throughout the chapters in this book.

Chapter 1 introduces the fundamentals of creating Windows Store applications and getting setup. It then shows some of the "quick win" functionalities available such as the ability to launch the Maps app that comes preinstalled in Windows 8. This is followed by an overview of the Bing Maps platform and how to create a Bing Maps account and key which you will need for most samples in this book.

Chapter 2 dives into the sensors and location platform that is built into Windows 8.

Chapters 3 and 4 introduces the Bing Maps for Windows Store Apps SDK's and how to use the most commonly used features. These chapters are broken up such that chapter 3 covers the JavaScript Bing Maps SDK and chapter 4 covers the native Bing Maps SDK.

Chapter 5 and 6 shows you how to make use of all the functionalities available in the Bing Maps REST services and the Bing Spatial Data Services.

Chapter 7 dives into the world of spatial data and shows how to integrate some of the more common spatial data formats into your Windows Store app.

Chapter 8 shows how to create a simple app that allows you to draw shapes on the map.

Chapter 9 shows you how to create an augmented reality app using several sensors in your device and a rear facing camera.

Chapter 10 shows how to create a reusable and templatable compass control.

Chapter 11 shows how to develop cross platform apps using both the JavaScript and Managed programming languages.

Throughout this book I have links to relevant documents, tools and resource. Some links are shorter and easier to type out than others. For the longer and harder to type links I have used a tool to shorten them so as to make them easier to manually type in a browser.

Prerequisites

To develop location based Windows Store applications you will need the following development environment installed:

- 1) **Windows 8.1:** If you are not already using Windows 8.1, download and install it. Get it here: <http://bit.ly/13GSDRk>
- 2) **Visual Studios 2013:** Download and install Visual Studios 2013 and the Windows Store SDK. Go to <http://bit.ly/INt02W> and select the "Download the tools and SDK" download. This download gives you Microsoft Visual Studio Express 2013 for Windows 8.1, Blend for Microsoft Visual Studio 2013 for Windows 8.1, and the Windows 8.1 software development kit (SDK), and project templates for creating new Windows Store apps.
 - Visual Studio gives you everything you need to create code, debug, localize, package, and deploy a Windows Store app.
 - Blend is another tool you can use to create Windows Store apps. It provides effective support for designing a great-looking user interface for your Windows Store app.

Visual Studio and Blend were designed to work together. You can move seamlessly back and forth between them to develop your app. If you wish to create cross platform apps as covered in Chapter 12, install the Windows Phone 8 SDK while installing Visual Studios.
- 3) **Bing Maps Windows Store Apps SDK:** Download and install the Bing Maps Windows Store Apps SDK for Windows 8.1: <http://bit.ly/17QDEla>

As you journey through this book, we will pick up some additional tools along the way. All of the additional tools used in this book are freely available, and details on how to obtain those tools are contained in the relevant chapters.

Design Tip

Creating a functional app is the goal most, if not all developers have. Unfortunately for many apps this is as far as it ever makes it. Creating a nice, engaging user experience is just as important as making the app functional. If the app doesn't look good then people are less likely to use it. Before launching your app have friends take a look at it and provide some feedback. If your friends wouldn't download it, you can't expect anyone else to. In this book I've included some design tips on how to create a better looking app that will keep your users coming back. Also, take a look at the design guides on creating Windows Store Apps: <http://bit.ly/OsK7Rt>

Downloading the code

This book contains numerous code samples that demonstrate how to create location based Windows Store Apps. All the samples are available for download in several different development languages including JavaScript, C# and Visual Basic from the MSDN code samples here: <http://bit.ly/1cfOtk2>. Here is a list of all the code samples covered in this book.

Chapter	Name	Description
1	Chapter1	An introduction to creating a Windows Store app. This app makes use of protocol activation to launch the built in maps app in Windows 8.
2	SensorsAndLocations	This app shows how to use all of the sensor and location APIs available to Windows Store apps
3 / 4	BingMapsIntro	This app demonstrates how to make use of all the key features in the Bing Maps SDK for Windows Store apps.
4	DraggablePushpins	This code sample shows how to create a draggable pushpin control for use in C# and VB apps.
5	BingMapsShareCharm	This app shows how to share maps via the share charm. Maps are shared in two ways; with a link and with an embedded static map image.
5	GeocodingSearchCharm	This app shows how to enable geocoding via the search charm.
6	BoundaryFinder	This app shows how to use the Bing GeoData API to search for and display boundaries on the map.
6	PointOfInterestMap	This app shows how to use the point of interest data sources in the Bing Spatial Data Services.
7	SpatialDataViewer	This app shows how to make use of the Microsoft.Maps.SpatialToolbox library to import several different spatial data file types into the Bing Maps Windows Store SDK.
8	MapPaint	This app shows how to use touch and mouse events to draw data on the map.
9	SimpleAR	This app shows how to use the compass and orientation sensors with a rear facing camera to create an augmented reality app that shows nearby points of interest.
10	Compass	This app shows how to create a reusable and templatable compass control for displaying the readings from the compass sensor.
11	CrossPlatformMaps	This code sample shows how to reuse code to make cross platform apps targeting Window 8, Windows Phone 8, and Web (JavaScript sample).

Contacting the author

If you have any questions or comments you can e-mail the author directly at ricky_brundritt@hotmail.com. Alternatively, Ricky can often be found trolling the Bing Maps forums (<http://bit.ly/111DXf2>) for questions to answer. You can also find many useful resources on Ricky's Bing Maps blog at <http://rbrundritt.wordpress.com>.

Recommended Reading

As hard as I might try I can't possibly fit everything there is to know about creating location intelligent applications. Here are a few resources that I have found very useful and recommend.

Books:

- Beginning Spatial with SQL Server 2008, Alastair Aitchison: <http://bit.ly/1gN86Gi>
- Pro Spatial with SQL Server 2012, Alastair Aitchison: <http://bit.ly/1e5jcmh>
- Microsoft Mapping, Ray Rischpater and Carmen Au: <http://bit.ly/1fZ6aZw>
- Programming Windows Store Apps with HTML, CSS, and JavaScript, Kraig Brockschmidt (free eBook): <http://bit.ly/1bGkX7e>

Blogs:

- Alastair Aitchison: <http://alastaira.wordpress.com/>
- Bing Maps Blog: <http://binged.it/HOiv5l>
- Hannes's Virtual Earth: <http://jkebeck.wordpress.com/>
- Maps for the Web: <http://www.web-maps.com/gisblog/>
- Pedro's Tech Mumbblings: <http://build-failed.blogspot.co.uk/>
- SharpGIS: <http://www.sharpgis.net/>

Microsoft Resources:

- Bing Maps Documentation: <http://bit.ly/1fDE6JH>
- Windows Dev Center: <http://bit.ly/1arxX1i>
- Bing Maps MSDN Code Samples: <http://bit.ly/1aViglT>

Chapter 1: Getting Started

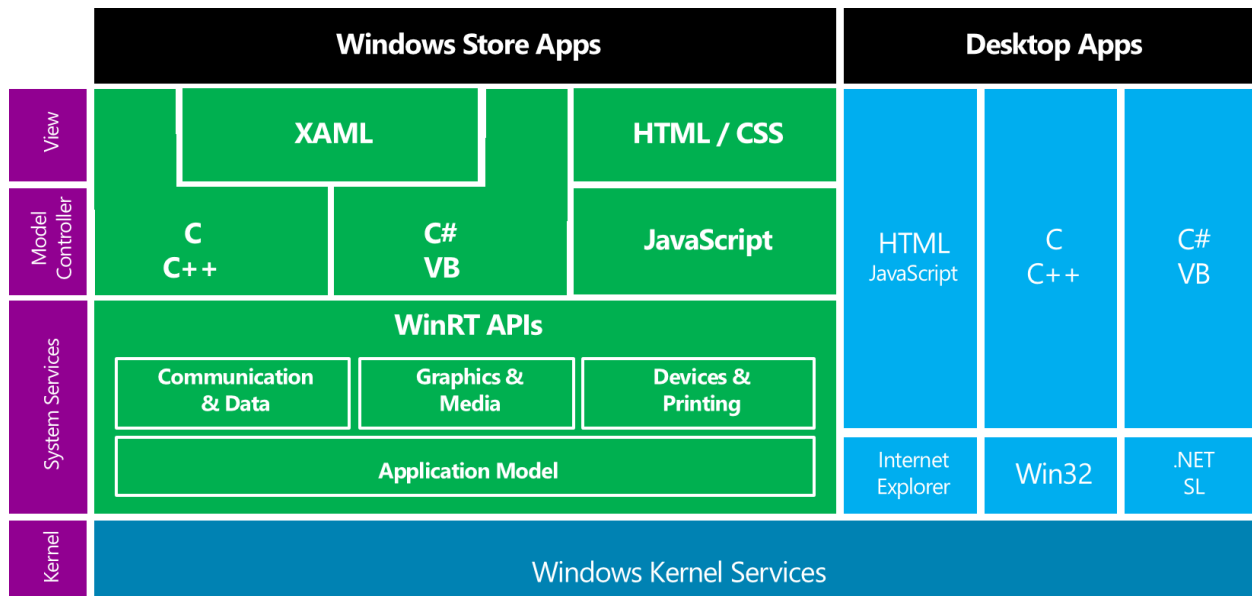
Introduction to Windows Store Apps

With Windows 8 we have the ability to target devices that use the x86 and x64 processors that we are used to using in previous versions of Windows. In addition to this we also have the ability to target devices that use ARM based processors too by using a special version of Windows 8 called Windows RT. This is important to know when developing applications for Windows 8 as traditional Win32 desktop applications will continue to work on x86 and x64 devices, but not on Windows RT. You can continue to develop Win32 based desktop applications however you will be limiting your application in a lot of ways. The biggest limitation being unable to share your application through the Windows Store which is the primary way users of Windows 8 install new applications. You would also be limiting your application to x86 and x64 based devices as Win32 applications are not supported on ARM based devices.

Windows Store apps are new to Windows 8 and are built on top of a framework called the Windows Runtime, also known as WinRT for short. Windows Store apps, unlike Win32 apps can work with all versions of Windows 8, including Windows RT. The goal behind the Windows Store framework is to provide developers with access to native capabilities within the Windows Kernel using a number of programming languages including JavaScript. The WinRT APIs are implemented using a low-level structure and then projected into different languages such as JavaScript, C#, Visual Basic, and C++. The WinRT API is projected such that it looks and feels natural to developers familiar to those languages.

Since Windows Store apps are all built on top of one common API (WinRT) this makes it possible to create a special Windows Store library called a Windows Runtime Component that can be referenced by any Windows Store application regardless of the programming language. This makes it easy to create highly reusable libraries for Windows Store apps. In addition it also gives us the ability to create very powerful applications regardless of the development language being used. Take for example you have an intensive computational task in a JavaScript application. To make your application much faster you could create a Windows Runtime Component library using managed (C# or Visual Basic) or native (C++) code which would be able to process this task much more efficiently than JavaScript can.

In the following diagram you can see the two different application development paths available in Windows 8. On the left is Windows Store Apps which are built on top of the WinRT APIs and on the right traditional Desktop applications which are built on top of the Win32 API.



Getting a developer license

The first time you run Microsoft Visual Studio 2013, you will be prompted to obtain a developer license. Developer licenses are free for Windows 8 and lets you install, develop, test, and evaluate Windows Store apps before having them certified by the Windows Store. By default, developer licenses that you acquire by using a Microsoft account must be renewed every 30 days. If the license expires (or you remove it) and you try and to run an uncertified Windows Store app you will be prompted to log into your Microsoft account and renew your license which is as easy as a click of the mouse. If have a valid developer license and try to run an uncertified Windows Store app you will receive an error (DEP0100) when you build or deploy the application in Visual Studios.

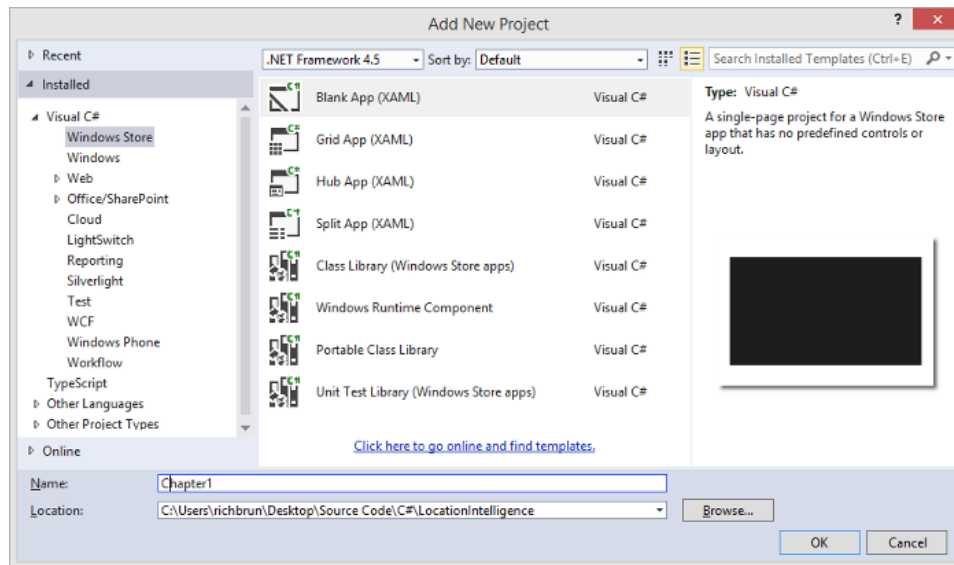
Before your app can be added to the Windows Store you must first package it and get it certified. If the app hasn't been certified it can't run on a Windows device unless a developer license is installed on the machine. The license is provided on a per-machine basis and for a fixed amount of time. After the developer license on your machine expires, you won't be able to run uncertified apps unless you renew the license.

Caution: By having a developer license, you can run Windows Store apps that haven't been tested and certified by the Windows Store. Running uncertified apps which are not protected by the certification process may put you at a higher risk of being infected by a virus or malware. If you acquire and run Windows Store apps from sources other than the Windows Store, take the same precautions you normally do when acquiring desktop apps from the web.

Creating a Windows Store App

To create a Windows Store app you need Windows 8.1, Visual Studio 2013 and the Windows Store SDK. If you haven't already installed these refer back to the Prerequisites section in the Introduction.

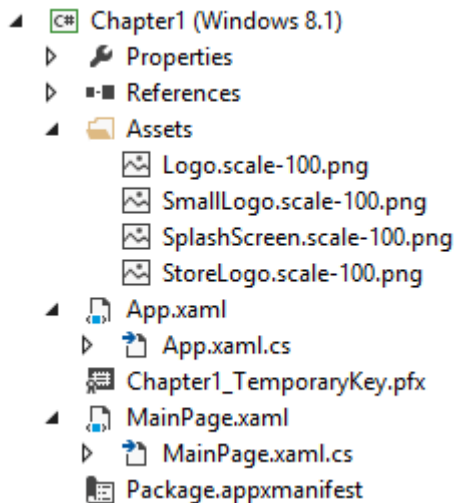
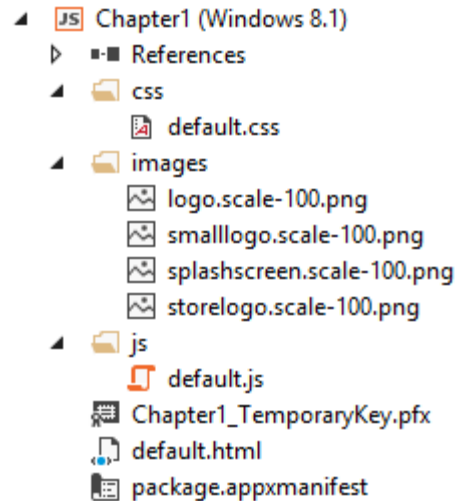
Once your development environment is setup you can open up Visual Studio and create a new Windows Store Project. To do this go to **File > New > Project** and then in the left pane, expand **Installed > Templates** select your preferred development language and pick the **Windows Store** template type.



Once selected you will be presented with a list of templates you can use to create Windows Store projects. The four project templates for creating Windows Store Apps are defined as the following:

Template	Description
Blank App	A project with nothing in it. This is a good starting point if you want to create a full screen app that doesn't require a template. This is also a good starting point for games and apps that fill the screen with a map or other single page user interface.
Grid App	The Grid app template consists of a home page that shows a list of groups. When a group is selected the app opens a group details page, which displays a list of the items on the right side. If the user selects an item from the home or details page a full-page view will open with the details for the selected item. This template is a good starting point for shopping apps, RSS readers, and photo or video apps.
Hub App	A three-page project for a Windows Store app that uses a Hub control. It displays content in a horizontally panning view and visually engages users by providing a variety of ways to access content. The main Hub page provides different sections to represent content such as new items, highlighted items, and categories of items.
Split app	The Split app template is a two-page project for a Windows Store app that navigates among grouped items. The first page displays a grid of groups. When a group is selected a second split-view page is displayed with a list of items within the group displayed on the left and details of the select list item on the right. This template is a good starting point for news readers, e-mail and messaging apps.

To start, select the Blank App template and give it a name of **Chapter1** then press OK. Depending on the development language you choose you will find that the Blank App template generated a Visual Studio project with one of the following structures.

**C# or Visual Basic App****JavaScript App**

Here is a description of the files in a C# or Visual Basic project.

File	Description
App.xaml App.xaml.cs/.vb	XAML and code files for the app.
Logo.scale-100.png SmallLogo.scale-100.png	A set of large and small logo images to display in the start screen.
MainPage.xaml MainPage.xaml.cs/.vb	A start page and an accompanying code file that runs when the app starts.
Package.appxmanifest	The manifest file that describes the app (its name, description, tile, start page, and so on) and lists the files that your app contains.
SplashScreen.scale-100.png	A splash screen to show when your app starts.
StoreLogo.scale-100.png	An image to represent your app in the Windows Store.

In the **Chapter1** project open the **MainPage.xaml** file and add a **StackPanel** with a margin of 40. Then add a **TextBlock** inside the **StackPanel** and set the Text property to "Chapter 1" and set the style to the **HeaderTextBlockStyle**. Putting this all together the XAML looks like this.

```
<Page
  x:Class="Chapter1.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:Chapter1"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d">

  <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel Margin="40">
      <TextBlock Text="Chapter 1" Style="{StaticResource HeaderTextBlockStyle}"/>
    </StackPanel>
  </Grid>
</Page>
```

The JavaScript project has a number of similar files. The key difference is that there is no XAML based files. Instead there is a **default.html** file that is the main entry point into the application. This file then references a JavaScript file called **default.js** and a CSS file called **default.css**.

Tip: For very simple applications in this book I may add JavaScript and CSS styles inline inside the HTML file to keep the steps simple. It is a best practice to separate your HTML, CSS and JavaScript into separate files. Not only does this keep your code clean but it also allows your application to make use of byte-code caching. Byte-code caching is a technique that creates byte-code for each JavaScript file once, rather than recreating the byte-code each time it launches the app. This is very similar to compiling code ahead of time.

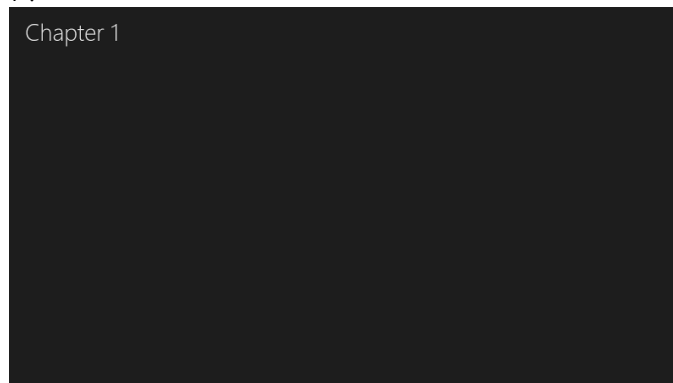
Open the **default.html** file and update it with the following code. This code adds a **div** and a **header** that says "Chapter 1" to the body of the document.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Chapter1</title>

  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
  <script src="//Microsoft.WinJS.2.0/js/base.js"></script>
  <script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

  <!-- Chapter1 references -->
  <link href="/css/default.css" rel="stylesheet" />
  <script src="/js/default.js"></script>
</head>
<body>
  <div style="margin:40px;">
    <h1>Chapter 1</h1>
  </div>
</body>
</html>
```

At this point, we created a very simple app. If you want to see what it looks like, press F5 to build, deploy, and start the app. A default splash screen appears first. When it disappears, the app will then appear with a black screen and white text saying "Chapter 1".



There is no button or command to close the app. You can close the app in a couple of different ways. One method is to press **Alt+Tab** to return to Visual Studio, then click **Debug > Stop debugging**.

Updating the app manifest

You can use the Manifest Designer in Visual Studio to edit the app manifest file that describes your app package. The app manifest file is present in Windows Store apps written in all languages. The Manifest Designer has six tabs:

Tab Name	Description
----------	-------------

Application	Used to configure application settings, such as the display name of the app, initial orientation, and default culture.
Capabilities	Specifies system features or devices that your app can use, such as Internet access, current location, and music library access.
Content URIs	Specify URIs that your app either can or can't access.
Declarations	Add declarations for app contracts, like search and share target contracts, and specify their properties.
Packaging	Set properties that identify and describe your package when it is deployed.
Visual Assets	Used to configure UI settings, such as the logos and splash screen.

To open the Manifest Designer, double-click the **Package.appxmanifest** file in your project, or right-click the file and click View Designer. The **Application** tab of the Manifest Designer is shown here:

Application Visual Assets Capabilities Declarations Content URIs Packaging

Use this page to set the properties that identify and describe your app.





Display name:

Entry point:

Default language: [More information](#)

Description:

Supported rotations: An optional setting that indicates the app's orientation preferences.

☐  Landscape
 ☐  Portrait
 ☐  Landscape-flipped
 ☐  Portrait-flipped

Minimum width: [More information](#)

Notifications:

Toast capable:

Lock screen notifications:

Tile Update:

Updates the app tile by periodically polling a URI. The URI template can contain "{language}" and "{region}" tokens that will be replaced at runtime to generate the URI to poll.

[More information](#)

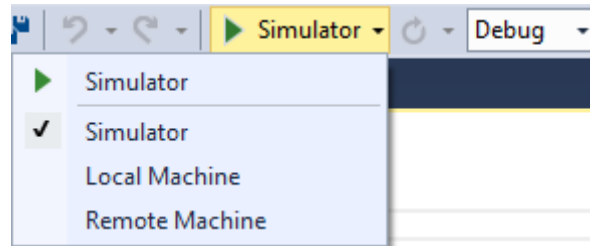
Recurrence:

URI Template:

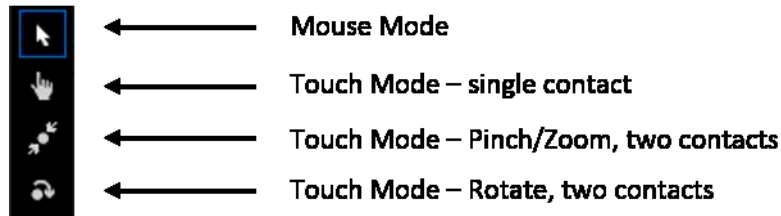
Using the Windows Store App Simulator

In Visual Studio you have the option to test your app in a simulator. The simulator provides an environment in which you can design, develop, and debug Windows Store apps. Using the simulator you can simulate touch events, rotate the device, change the screen size and resolution, provide location information, and capture screenshots. The simulator is a great tool for testing your apps but you should also test your app on a proper Windows device before submitting it to the Windows Store.

To run your app in the simulator, select the Simulator option from the drop-down list next to the start debugging button on the debugger toolbar.



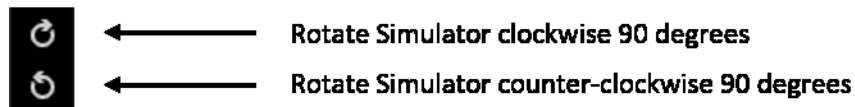
The touch simulator enables developers to test and debug how an application reacts to different touch gestures. The simulator does all the hard work of converting mouse events into touch events received by the running application. This is especially useful for developers who do not have a touch screen monitor. The following is what these mouse/touch options look like on the simulator.



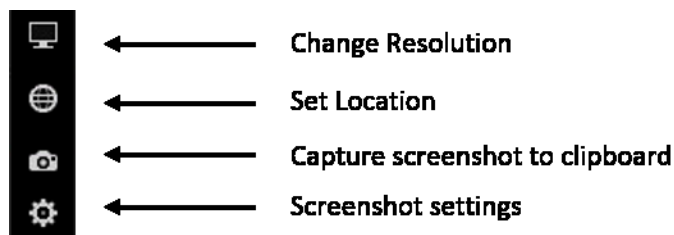
Using these controls you can emulate the following touch gestures:

- Tap (single contact)
- Hold (single contact)
- Double Tap (single contact)
- Pan/Swipe (single contact)
- Flick/Inertia (single contact)
- Pinch/Zoom (2 contact points)
- Rotate (2 contact points)

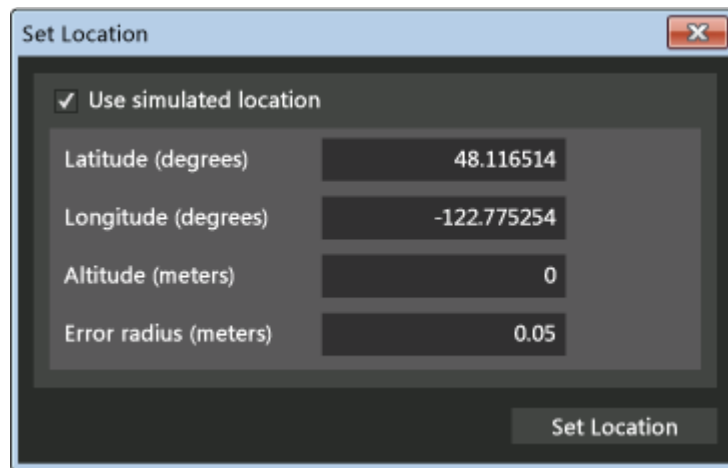
All Windows Store apps should support Landscape and Portrait modes. Without access to a device that supports changes in orientation it can be hard to test that your application layout works in both Portrait and Landscape mode. You can switch the device orientation between portrait and landscape by rotating the simulator 90 degrees in any direction.



Windows devices support several different screen sizes and Resolutions. You can use the simulator to see how your app looks using different screen resolutions. The "Change Resolution" button on the Palette allows you to choose a resolution to use in the simulator. The availability of resolutions in the Palette depends upon the supported resolutions of your video card.



For location-aware apps you can change the location of the device using the “Set Location” button. This will cause a dialog box to open where you can specify the Latitude, Longitude, Altitude, and an error radius. The error radius value indicates the accuracy level of the latitude and longitude coordinates, specified in meters.



When you submit an app to the Windows Store, you must include screenshots of the app. You may also want to take screenshots for sharing or presentations. To create screenshots of your app from the simulator, choose the “Capture screenshot to clipboard” button. The screenshot is saved at the current resolution of the simulator. You can also use the screenshot settings to specify a folder to save screenshots to also.

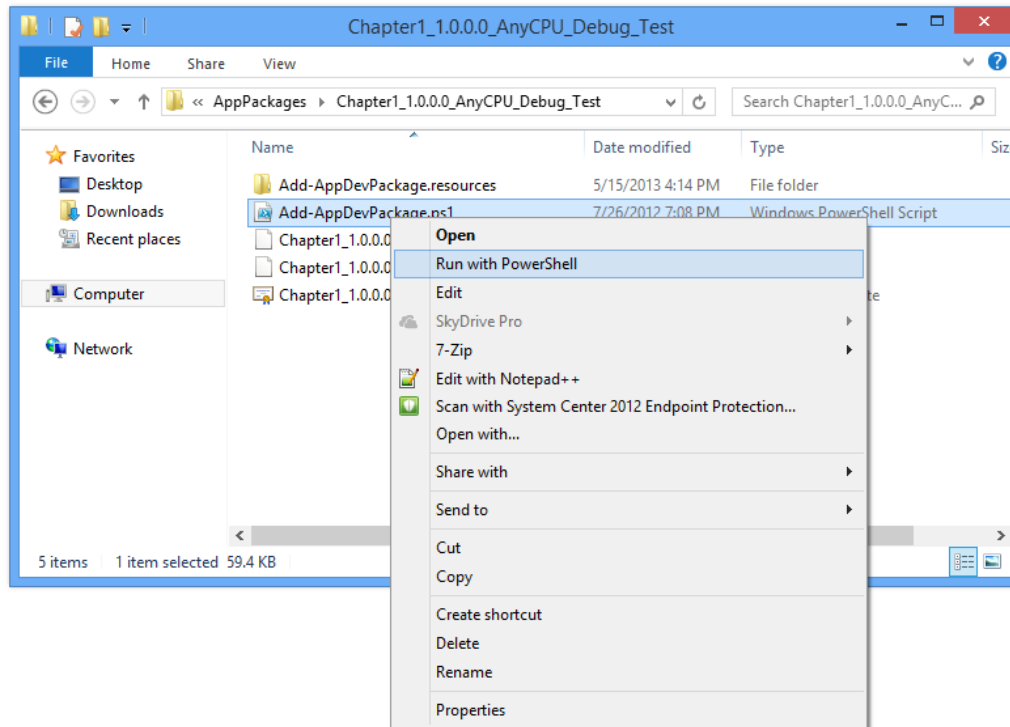
Important Note: At the time of writing this book the simulator does not provide any tools for simulating sensors. If your application uses sensor you will need to test your app on a device that has the required sensors.

Side Loading a Windows Store app

It’s a good practice to test your app on more than one device. An app that runs great on your development computer may not run as smoothly on other devices. This is especially important if you plan to make your application available to devices that use ARM based chips. You can easily deploy your app to another device using a process called side loading. Side loading simply means installing a Windows Store app directly to a device without publishing it and downloading it from the store.

Before you can side load an app you first need to package your app. To do this right click on your project and select **Store > Create App Packages**. The Create App Packages wizard will open and ask you if you want to create a package to upload to the Windows Store. Select **No** then press the **Next** button. This allows you to create a Windows Store app without registering and submitting it to the store. On the next screen select the **Create** button. This will create the package for you and will then display a link to the file folder the package was created in. Navigate to this directory. In this directory you will find a folder that has your application name and version information. Copy this folder to your other device using a USB drive or zipping the folder and emailing it to yourself and download load it to your other device.

Now that you have the app package on your other device open the folder. Inside you should see a number of files. One of the files is a Windows PowerShell Script. Locate this file and right click on it (or press and hold if using touch). In the context menu that opens click on the **Run with PowerShell** command.



Follow the prompts that come up on the screen. The first time you do this you will be prompted to obtain a developer license for this new device. Once this is done you won't see this prompt again until the license expires, at which point you can easily renew it. Once this is done you can go to the Start screen and at the end you should see your app.

Linking to the Maps app

In Windows Store apps we have the ability to use Uri's to launch other applications that are installed on a user's computer from your own app to perform a specific task. This is known as protocol activation and can be achieved by using the **Windows.System.Launcher** class. For example, if you want to allow the user to send an email to a contact in your app you can use the mailto: Uri to launch the user's default e-mail app. Windows 8 comes prepackaged with a Maps application that makes use of Bing Maps. This application exposes a set of URIs that can be used to pass in information to display on the map. This is a great way to quickly add mapping functionality to your application without having to have any knowledge of how to use Bing Maps.

Tip: In addition to using protocol activation to launch other applications, you can also expose a set of Uri's that can be used by other apps to launch yours. For instance, if you created a real estate application you may want to allow other applications to pass in a search location into your app to find properties. This is a great way to get more people using your app. You can find more information on how to handle protocol activation here: <http://bit.ly/1936HDg>

There are some benefits to launching the Maps app rather than developing the mapping functionality in your application. The biggest benefit is that this will usually require less development effort by you. Another benefit is that you will be sending your users into a mapping application that they are likely to be already familiar with. However, the following are potential limitations that you should take into consideration:

- **You are sending your users away:** By launching another application you are sending your users away from yours. In most situations you will want to keep the users in your application for as long as possible. This is not that much of an issue if both apps work well in snapped mode as the second app will usually appear beside your app.
- **Limited to small amounts of data:** Uri's are limited to how long they can be (usually 2,083 characters). This means that you will only be able to pass in a small number of data points (approximately 25) to display on the map before your Uri gets too big.
- **No customizations:** One of the first things many developers want to do is create a pushpin that uses a custom icon. The Maps app doesn't expose any functionality for doing this.
- **Limited Functionalities:** By launching another application you are not limited by the functionalities that are in that application, but by the functionalities that are exposed via the Uri. Some applications may not expose all the functionalities you want or need.

If you want to create more advance mapping functionalities then you will want to use the Bing Maps SDK for Windows Store Apps.

Creating a Uri to the Maps App

The Maps app has a number of different parameters that can be used to create an Uri that launches the app and performs a specific function. The base Uri for the Maps app is **bingmaps:** Simply passing this in as your Uri will open the Maps app as normal. You can add a query string that makes use of the following parameters to create a unique Uri.

Parameters	Definition	Details
cp	Center Point	Defines where the center of the map should be. Latitude and longitude values should use the following format: Latitude~Longitude Where both values are expressed in decimal degrees. Example: cp=47.677~-122.122
bb	Bounding Box	A rectangular area that specifies the bounding box expressed in decimal degrees, using a tilde (~) to separate the lower left corner from the upper right corner. Latitude and longitude for each are separated with an underscore (_) as in the following format: bb=southLatitude_eastLongitude_northLatitude_westLongitude Valid longitude values are between -180 and +180 inclusive. Valid latitude values are between -90 and +90 inclusive. Example: bb=39.719_-74.52~41.71_-73.5
where	Location	Search term which is a location, landmark or place. Example: where=Paris
q	Query Term	Search term for local business or category of businesses. Example: q=coffee
lvl	Zoom Level	Defines the zoom level of the map view. Valid values are 1-20 where 1 is zoomed all the way out. Example: lvl=15
sty	Style	Defines the map style. Valid values for this parameter include:

		<ul style="list-style-type: none"> • a: Display an aerial view of the map. • r: Display a road view of the map. (default) <p>Example: sty=a</p>
trfc	Traffic	<p>Specifies whether traffic information is included on the map. Valid values are 0 (default, not included) or 1 (included).</p> <p>Example: trfc=1</p> <p>Traffic Coverage information: http://bit.ly/130ytgc</p>
rtp	Route	<p>Defines the start and end of a route to draw on the map, separated by a tilde (~). Each of the waypoints are defined by either a position (pos) using latitude and longitude separated using an underscore (_) or an address (adr) identifier.</p> <p>rtp=adr.Place~pos.latitude_longitude</p> <p>A complete route contains exactly two waypoints. For example:</p> <p>rtp="A"~"B"</p> <p>However, it is acceptable to specify an incomplete route so as to populate the To or From field. For example:</p> <p>rtp="A"~ or rtp=~"B"</p> <p>Example: rtp=adr.Seattle,%20WA~pos. 47.640131_-122.129732</p>
collection	Collection	<p>Collection of entities to be added to the map and list. Collections can have a name and up to 25 points.</p> <p>Example: collection=name.MyCollection~point.40.726966_-74.006076_Pin%20Title~ point.43.678_-122.435_My%20Pin</p>
point	Point	<p>Specifies a point to add to a collection. For points, the value includes the latitude, longitude, and title (max of 255 characters), each separated by an underscore (_):</p> <p>collection=point.latitude_longitude_title</p> <p>If the title you specify contains an underscore, make sure the underscore is double encoded as %255F.</p> <p>If a point is defined without a name, the title will be "Custom pin"</p> <p>Example: collection=point.43.678_-122.435_My%20Pin</p>
name	Name of collection	<p>Name of the collection. If a name is not provided and there is more than one entity in the collection, the default name is "Custom pins".</p> <p>Example: collection=name.MyCollection</p>

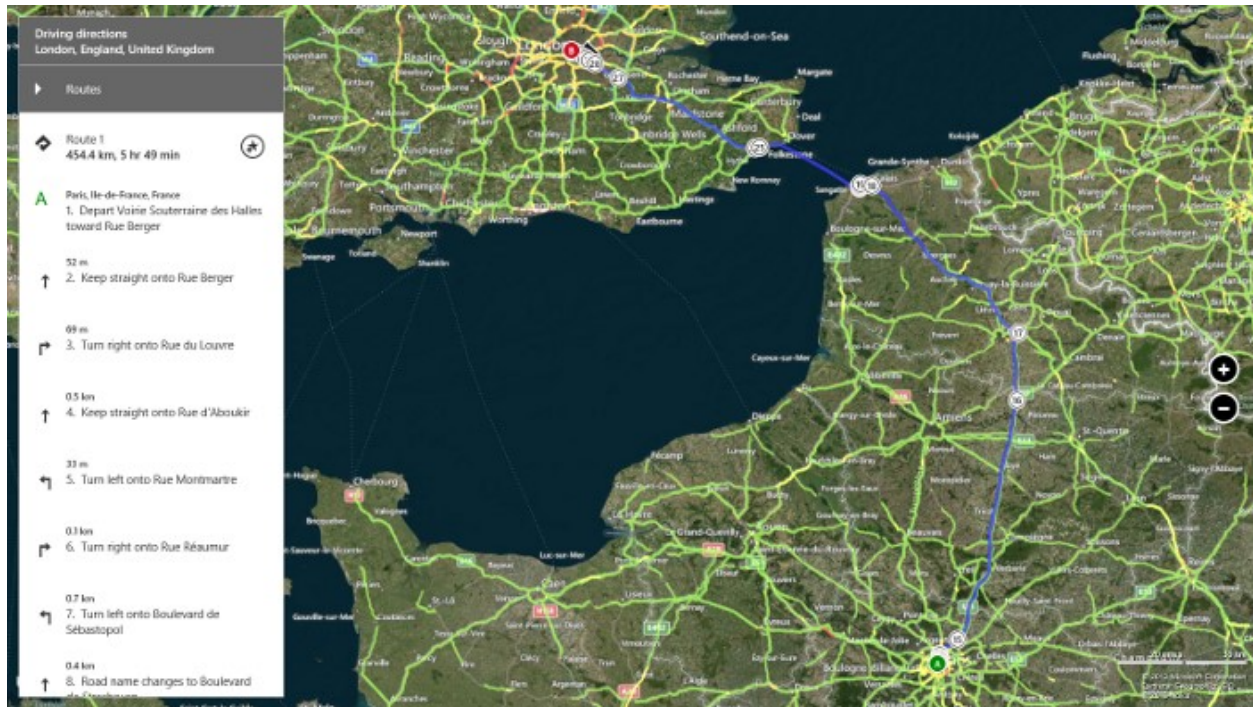
You can now start creating Uri's that load the map in different ways. If you wanted to load the map over New York, with a zoom level of 10 you could create the following Uri:

bingmaps:?lvl=10&where=New%20York

Going a bit further you could use the following Uri to display an aerial map with traffic data overlaid and a route from Paris to London:

bingmaps:?rtp=adr.Paris~adr.London&sty=a&trfc=1

You can easily test these Uri's out by putting them in the address bar of a browser. If you are using Windows 8 you will be prompted with a message asking if you would like to allow the Maps app to open. If you allow it the Maps app will load and process your query. Using the last example Uri you will end up with a map like this:



Tip: The maps app only supports passing in latitude and longitude coordinates with 6 decimal places or less. Adding more decimal places can cause the maps app to load in incorrect locations. That said 6 decimal places has an accuracy of 10 centimeters which is more than accurate enough for most applications. To ensure that this doesn't cause any issues you can add a string formatter that limits the numbers to 6 digits (i.e. {0:N6}).

Launching the Maps App

Now that we know how to create an Uri that will launch the Maps App we can now look at how to implement these Uri's into your app. The **Windows.System.Launcher** API can be used to launch web pages or files and includes the following methods:

C#/VB Method	JavaScript Method	Description
LaunchUriAsync	launchUriAsync	Starts the default app associated with the Uri scheme name for the specified URI.
LaunchFileAsync	launchFileAsync	Starts the default app associated with the specified file.

To launch the Maps app you will want to use the **LaunchUriAsync** method. Open the **default.html** or the **MainPage.xaml** file and add the following code after the header.

HTML

```
<br />
```

```

<p>
  <button id="openMapAppBtn">Open Map App</button><br /><br />
  <button id="openMapAppNewYorkBtn">Show map of New York</button><br /><br />
  <button id="openMapAppRouteBtn">Show map of route from Paris to London</button>
</p>

```

XAML

```

<StackPanel Margin="0,30,0,0">
  <Button Content="Open Map App" Click="OpenMapApp_Click"/>
  <Button Content="Show map of New York" Click="OpenMapAppNewYork_Click"/>
  <Button Content="Show map of route from Paris to London" Click="OpenMapAppRoute_Click"/>
</StackPanel>

```

This will add three buttons to the app. The first button will open the map app, the second button will open the map app and look for New York, and the third button will open the map app and calculate a route from Paris to London. In order for these buttons to work the event handlers will need to be updated. Open the **default.js** or **MainPage.xaml.cs** file and update them with the following code.

JavaScript

```

(function () {
  "use strict";

  var app = WinJS.Application;
  var activation = Windows.ApplicationModel.Activation;

  app.onactivated = function (args) {
    if (args.detail.kind === activation.ActivationKind.launch) {
      if (args.detail.previousExecutionState !==
activation.ApplicationExecutionState.terminated) {
        //Initialize the map app links by adding the button event handlers
        initializeMapAppLinks();
      } else {
      }
      args.setPromise(WinJS.UI.processAll());
    }
  };

  app.oncheckpoint = function (args) {
  };

  app.start();

  function initializeMapAppLinks() {
    //Add click events to the Map App buttons
    document.getElementById("openMapAppBtn").addEventListener("click",
function() {
  openApp("bingmaps:");
}, false);

    document.getElementById("openMapAppNewYorkBtn").addEventListener("click",
function() {
  openApp("bingmaps:?lvl=10&where=New%20York");
}, false);

    document.getElementById("openMapAppRouteBtn").addEventListener("click",
function() {
  openApp("bingmaps:?rtp=adr.Paris~adr.London&sty=a&trfc=1");

```

```

    }, false);
}

function openApp(appUri) {
    // Create a Uri object from a Uri string
    var uri = new Windows.Foundation.Uri(appUri);

    // Launch the URI
    Windows.System.Launcher.launchUriAsync(uri).then(
        function (success) {
            if (success) {
                // URI launched
            } else {
                // URI launch failed
            }
        }
    ));
}
})();

```

C#

```

using System;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

namespace Chapter1
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
        }

        private void OpenMapApp_Click(object sender, RoutedEventArgs e)
        {
            OpenApp("bingmaps:");
        }

        private void OpenMapAppNewYork_Click(object sender, RoutedEventArgs e)
        {
            OpenApp("bingmaps:?lvl=10&where=New%20York");
        }

        private void OpenMapAppRoute_Click(object sender, RoutedEventArgs e)
        {
            OpenApp("bingmaps:?rtp=adr.Paris~adr.London&sty=a&trfc=1");
        }

        private async void OpenApp(string appUri)
        {
            try
            {
                // Create a Uri object from a Uri string
                Uri uri = new Uri(appUri);

                var success = await Windows.System.Launcher.LaunchUriAsync(uri);

                if (success)
                {
                    // Uri launched
                }
            }
            catch { }
        }
    }
}

```

```

        }
        else
        {
            // Uri launch failed.
        }
    }
    catch { }
}
}
}

```

Visual Basic

```

Imports System.Text

Public NotInheritable Class MainPage
    Inherits Page

    Private Sub OpenMapApp_Click(sender As Object, e As RoutedEventArgs)
        OpenApp("bingmaps:")
    End Sub

    Private Sub OpenMapAppNewYork_Click(sender As Object, e As RoutedEventArgs)
        OpenApp("bingmaps:?lvl=10&where=New%20York")
    End Sub

    Private Sub OpenMapAppRoute_Click(sender As Object, e As RoutedEventArgs)
        OpenApp("bingmaps:?rtp=adr.Paris~adr.London&sty=a&trfc=1")
    End Sub

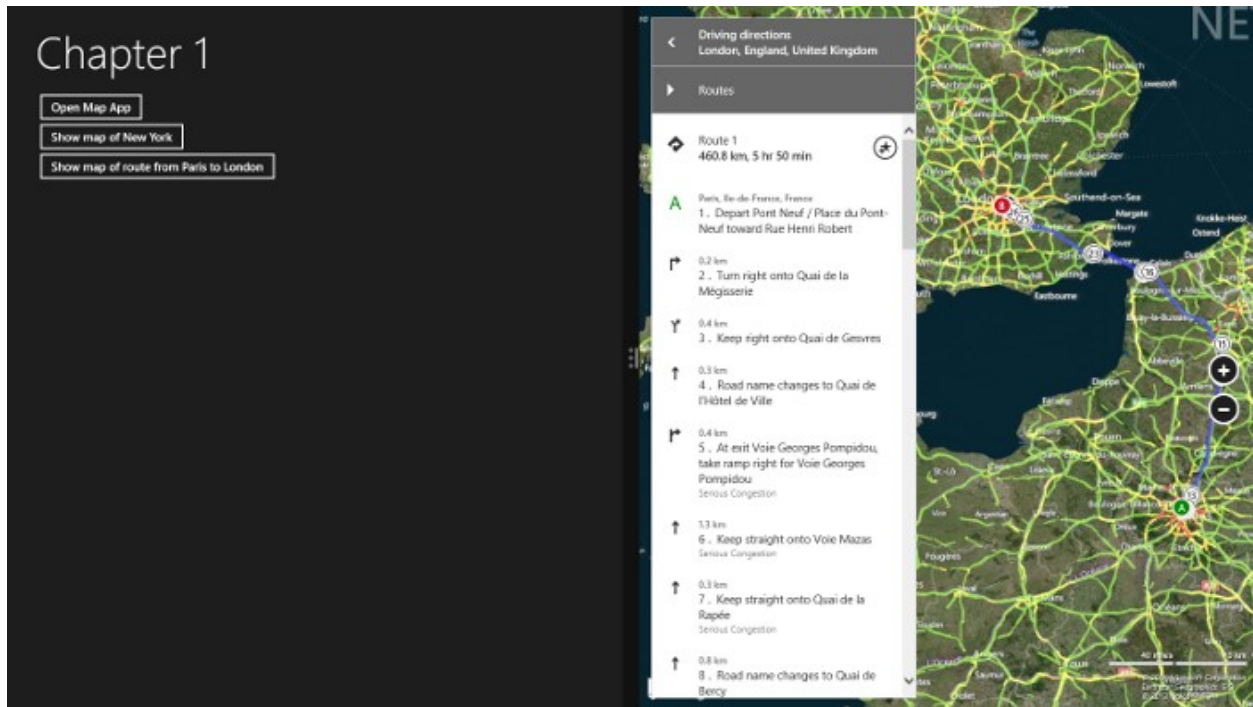
    Private Async Sub OpenApp(appUri As String)
        Try
            ' Create a Uri object from a Uri string
            Dim uri = New Uri(appUri)

            Dim success = Await Windows.System.Launcher.LaunchUriAsync(uri)

            If success Then
                ' URI launched
            Else
                ' URI launch failed
            End If
        Catch
        End Try
    End Sub
End Class

```

If you run the application (F5) and press one of the buttons in the app you will see the Map app launch beside your app in split screen mode like this.



Bing Maps Platform

The Bing Maps platform brings location data to life by making it easier to visualize, understand, and analyze. The rich imagery, quality geospatial data, and leading-edge technology of Bing Maps is used by developers around the world. The Bing Maps platform consists of a number of robust APIs and cloud based Services. In fact, it is one of Microsoft's first cloud based services and was used as a model on which Windows Azure was originally designed. Bing Maps is not only used in one of the key Windows 8 apps, the Map apps, but is also the main mapping platform used for developing location based Windows Store Apps. The Bing Maps platform is powered by several petabytes (1 petabyte = 1,000 terabytes) of data. Because of this you will always need an internet connection to pull in new map imagery or access any of its services. The following is a list of the most commonly used API's and Services provided by the Bing Maps platform.

Bing Maps API	Description	Target Platforms
Bing Maps Windows Store Native API	This is one of the newest Bing Maps APIs and allows you to add Bing Maps to C#, VB or C++ based Windows Store apps. This API makes full use of hardware acceleration which makes it ideal for high performance applications.	Windows Store Apps
Bing Maps Windows Store JavaScript API	This API is based on the Bing Maps AJAX V7 control which is ideal if you want to reuse an existing Bing Maps AJAX application.	Windows Store Apps
Bing Maps V7 AJAX control	This JavaScript API is one of the most universal mapping controls available. Not only is it supported on standard PC & Mac browsers, but it is also supported on all the major mobile platforms as well making it ideal for cross platform applications.	Web, Mobile
Bing Maps REST Services	This is an excellent service for performing tasks such as geocoding, reverse-geocoding, routing and static imagery. Being a REST-based service, it can be easily accessed from almost any development environment.	Windows Store Apps, Web, Mobile, Desktop

Bing Spatial Data Services	This is also a REST-based service that offer four key functionalities: batch geocoding, point of interest (POI) data, boundary data and the ability to store and expose your spatial data. This service is ideal for those who need a place to store their spatial data, or who need point of interest data in their application.	Windows Store Apps, Web, Mobile, Desktop
Bing Maps WPF control	The Bing Maps WPF control is an excellent control for creating traditional desktop-based applications.	Desktop

As you can see you have a number of different API's and Services to choose from. So how do you know which one to use? Here are a couple of tips:

- If you are targeting Windows Store apps then use the Bing Maps Windows Store API that uses the language you are most comfortable using or is required by your application. Chapters 3 & 4 goes into API's in detail.
- If you are creating a cross platform application (Windows Store Apps, Web, Mobile) than use the Bing Maps Windows Store JavaScript and the Bing Maps V7 AJAX controls. There are some slight differences between these two API's however these are easy to work with. We will discuss cross platform development using these API's in Chapter 11.
- If you are targeting Windows Phone 8 then use the built in Microsoft mapping control in the Windows Phone API for creating native apps
- If you are creating a traditional desktop application that will run on Windows 7 & 8 then use the Bing Maps WPF control. The WPF control can integrate into WPF and WinForm applications.
- If you require a place to host spatial data (i.e. points, polygons) then use the Bing Spatial Data Services. It will expose your data set as a spatial REST service which you can then easily tie into your application. We will look deeper into the Bing Spatial Data Services in Chapter 6.
- If you need to geocode, reverse-geocode, calculate routes, create static map images, and retrieve elevation or traffic data without loading an interactive map control then use the Bing Maps REST services. The benefit of the REST services is that they can be used independently of any of the map controls which means that if you don't need an interactive map then just connect directly to the services and save resources in your application. In Chapter 5 we will learn all about the Bing Maps REST services.
- If you need to geocode a large number of addresses then use the Bing Spatial Data Services.

You will likely only use one map control in your application but may find yourself using one or both of the services. Both the Bing Maps Windows Store Native and JavaScript API's are part of the Bing Maps Windows Store SDK. Since this book is on Windows Store apps we will be using the Bing Maps Windows Store SDK, the Bing Maps REST Services, and the Bing Spatial Data services in this book. If you have not already installed this SDK you can download it here: <http://bit.ly/11TLdei>.

Key Features

There are a large number of features available in Bing Maps. Below is a list of some of the more interesting ones.

Feature	Description
Road Maps	Road maps are one of the more basic types of maps offered in Bing Maps. This map style was designed to be more of a canvas for overlaying data such that the data will stand out better. In addition to that it has also been designed for those with color blindness in mind.
Aerial Imagery	This type of map overlays imagery that has been captured by planes and satellites pointing straight down. Bing Maps has some of the highest resolution imagery of this type available from any online map provider. In addition to that they also have one of the most up to date and complete data sets of this type of imagery as well.
Birdseye Imagery	Birdseye imagery is captured by low-flying aircraft at a 45 degrees angle. This type of imagery is typically captured from 4 different directions. This gives the user the ability to rotate the imagery around a location and offers better depth perception for buildings and

	geography. These images are typically much more detailed than the aerial views taken from directly above.
Ordnance Survey Maps	Bing Maps provides access to the Ordnance Survey (OS) 1:25,000 and 1:50,000 maps in the UK. The OS is one of the oldest and largest producers of maps with a focus on Great Britain. The OS produces the most detailed maps of Great Britain and include everything from roads to hiking trails. Bing Maps also has an agreement with the OS that allows you to overlay their data and any OS derived data onto Bing Maps applications.
Venue Maps	Venue Maps are often maps of indoor structures such as malls and airports, however venue maps are not just limited to this. Other types of venue maps include the layout of shopping districts, stadiums, race courses, and universities. Bing Maps has thousands of venues maps available around the world.
Directions	Bing Maps provides one of the fastest and most accurate route directions available by any online mapping provider. In addition to driving direction Bing Maps also provides walking and transit directions. Transit directions are not available everywhere. You can find the coverage information for transit directions here: http://bit.ly/12ARybp
Traffic data	Bing Maps provides both traffic incident data and traffic flow overlays in several countries around the world. You can find the traffic coverage information here: http://bit.ly/130Ytgc
Language support	Windows supports a large number of languages. Since Bing Maps is a key feature in Windows 8 it supports 116 languages for geocoding and routing.

Free Terms of Use

Bing Maps is an Enterprise platform that offers free terms of use to its users under certain conditions. At the time of writing this book, Windows Store apps can use the free terms of use if they meet the following requirements:

- Are publicly available to anyone through the Windows Store as a free or paid for app.
- Is not a business asset tracking application (requires an Enterprise license)
- Is not providing real-time navigation (automatically updating route instructions as a user's location changes).
- Adheres to the full Bing Maps terms of use.

Applications that meet these requirements are allowed to generate up to 50,000 transactions in any 24 hour period. This is a generous free usage limit and far exceeds the limits given by any other cloud based mapping platform. Transactions occur when the map loads and when using the Bing Maps services to perform operations such as geocoding and routing. I list detailing all the different types of transactions can be found here: <http://bit.ly/130UwmS>. It is also worth pointing out that these same Free Terms of Use also apply to mobile application that use Bing Maps.

Applications using the free terms of use are subject to be rate limited if they generate a high frequency of transactions. When rate limiting occurs an empty response will be returned by the Bing Maps services and a flag that indicates rate limiting has occurred will be in the header of the response. Rate limiting does not occur in applications that use an Enterprise Bing Maps key (requires a Bing Maps Enterprise license).

Note that the terms of use may change over time. You can access the current terms of use here: <http://www.microsoft.com/maps/product/terms.html>

Creating a Bing Maps account

One of the first things you will need to do before you can use the Bing Maps SDK is to create a Bing Maps account. The following steps outline how to create a Bing Maps account:

1. Go to the Bing Maps Account Center at <http://www.bingmapsportal.com> and click the "Create a Bing Maps Account" button.
2. Sign in using a Microsoft Account (formerly known as a Windows Live ID). If you do not have a Microsoft Account then use the sign up link on the sign in page.

3. After you sign in, provide the following information:
 - **Account name:** A friendly name that you can use to identify your account. (Required)
 - **Contact name:** The name of the account owner or someone that can be contacted when questions about this account arise. (Optional)
 - **Company name:** The name of the company using this account. (Optional)
 - **Email address:** A contact email address for this account. This address can be the Microsoft Account (Windows Live ID) used to login to the Bing Maps Account Center. (Required)
 - **Phone number:** A contact phone number for this this account. (Optional)
 - **Bing Maps API terms of use:** Review the terms of use and check the box to accept them. (Required)
4. After you enter the required information and click Save, your account details display.

Creating a Bing Maps key

Once you have a Bing Maps account you will need to create a Bing Maps key. A Bing Maps key is used to authenticate your requests to Bing Maps. You will need to use this every time you load a map and any time you call any of the Bing Maps services directly. The following steps outline how to create a basic Bing Maps key which you will use in the examples throughout this book.

1. Go to the Bing Maps Account Center at <http://www.bingmapsportal.com> and sign in using your Microsoft Account (Windows Live ID).
2. Select "Create or view keys" under "My Account".
3. In the **Create key** box on the My keys page, provide the following information for the application that will use the Bing Maps key:
 - **Application name:** The name of the application. This is for your own use so you can easily identify the purpose of the key. (Required)
 - **Application URL:** The URL of the application. This is also for your own reference. You can leave this blank. (Optional)
 - **Key type:** Select the **Basic** key type. The key and application types you choose determine your usage limits. Note **Trial** keys expire after 90 days and **Enterprise** keys require a Bing Maps license. (Required)
 - **Application type:** Select the **Windows Store App** application type.
4. Type the characters of the security code, and then click Submit. The new key will be displayed in the list of available keys under the form. (Required)

A Bing Maps key is a 64 bit encoded string which will look something like this:

```
Ah_C8OJJu8wnNX50rGHf_OYKonuhZ-CfLQ-kXS-4tl-QsTN9pkLPPfgKigwa
```

Make note of the key you created as you will need to include it in many of the code examples in this book. Copying it into a text file and saving it locally will save you having to sign into the Bing Maps portal to retrieve it.

The Bing Maps Tile System

Bing Maps provides a world map that users can directly manipulate to pan and zoom. To make this interaction as fast and responsive as possible, the map is broken up into a large number of images at different levels of detail. These images are often referred to as tiles as they are stitched together by the map control like a tiled surface to create a complete map. This technique is used by most online mapping providers. Tile layers are also often used as a way to render large amounts of data on maps without performance issues, as an image of rendered data is often a lot smaller and easier to handle than a lot of raw data. We will discuss this in more detail in Chapter 7.

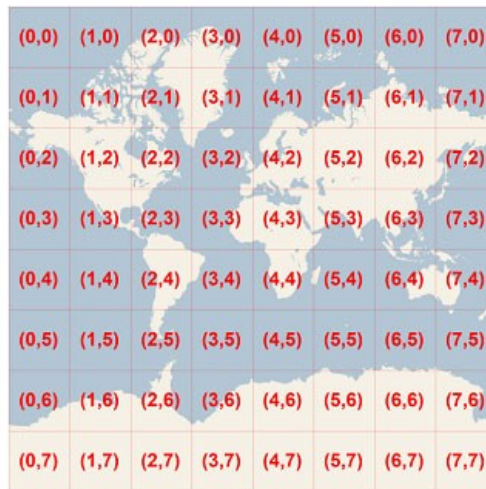
Bing Maps uses what's called a spherical Mercator projection. A projection is the mathematical model used to transform the spherical globe into a flat map. The Mercator projection stretches the map at the poles in order to create a nice square map. This significantly distorts the scale and area of the map but has two important properties that outweigh this distortion:

1. It's a conformal projection, which means that it preserves the shape of relatively small objects. This is especially important when showing aerial imagery, because we want to avoid distorting the shape of buildings. Square buildings should appear square, not rectangular.
2. It's a cylindrical projection, which means that north and south are always straight up and down, and west and east are always straight left and right.

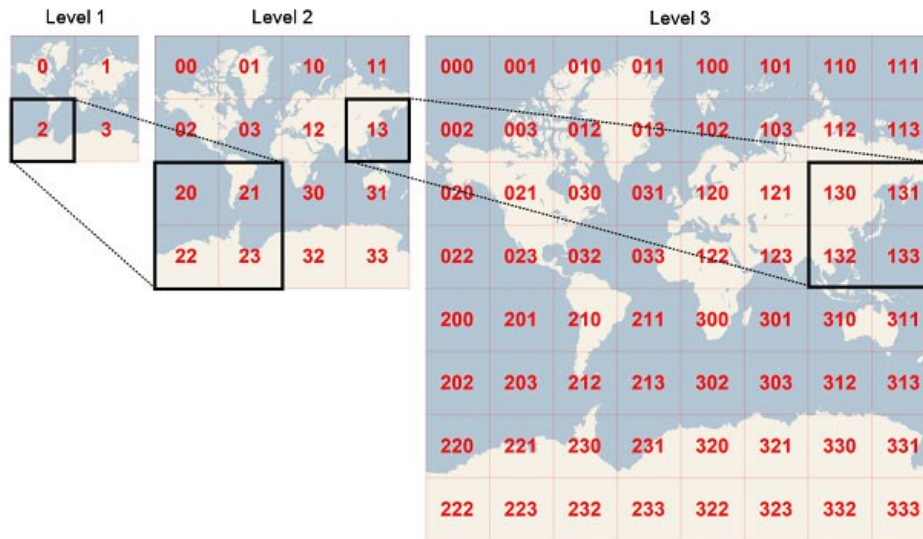
To optimize the performance of map retrieval and display, the rendered map is cut into tiles of 256 x 256 pixels each. As the number of pixels differs at each level of detail (zoom level), so does the number of tiles:

$$map_{width} = map_{height} = 256 * 2^{Zoom\ Level}$$

Each tile is given an XY index ranging from (0, 0) in the upper left to (2^{level}-1, 2^{level}-1) in the lower right. For example, at level 3 the tile index ranges from (0, 0) to (7, 7) as follows:



To optimize the indexing and storage of tiles, the two-dimensional tile XY indices are combined into one-dimensional strings called quadtree keys, or "quadkeys" for short. Each quadkey uniquely identifies a single tile at a particular zoom level, and it can be used as a key in a database index. To convert a tile index into a quadkey, the bits of the Y and X components are interleaved, and the result is interpreted as a base-4 number (with leading zeroes maintained) and converted into a string. Quadkeys have several interesting properties. First, the length of a quadkey (the number of digits) is equals to the zoom level of the corresponding tile. Second, the quadkey of any tile starts with the quadkey of its parent tile (the containing tile at the previous level). As shown in the example below, tile 2 is the parent of tiles 20 through 23, and tile 13 is the parent of tiles 130 through 133.



You can find additional details about the Bing Maps tile system and the mathematics behind it here:

<http://bit.ly/15RcScZ>

Design Tip: Segoe UI Symbol Buttons

Creating text buttons is easy to do but is often over used and doesn't really look all that great. Take for example this simply button.

HTML

```
<button>Add Pushpin</button>
```

XAML

```
<Button Content="Add Pushpin"/>
```

This ends up looking like the following image.



Sometimes it may be better to have an image as a button instead. This can also be easily done simply by adding a click or touch event to an image. Unfortunately there are a couple of disadvantages to this approach. First you lose the nice press states (shading) that come with buttons that let the user know the button is pressed. You could work around this and create your own pressed states but this can get very complicate very quickly. The second disadvantage is that if you want to change the color of the image you have to go through and edit it manually. This is a pain when trying to handle high contrast mode as you will often end up having to create two images. As a result it doesn't take long before your app has a ton of images in it just for buttons which may make it hard to manage. Also, most developers write code and are not graphic designers, so creating nice looking images could be a hurdle in itself. If only there was a bunch of pre-made images you could use that were easy to customize in code.

Segoe UI Symbol is a type of font style available in Windows Store apps. This font style provides a ton of really cool looking symbols that can be easily used as images for buttons. Since this is just a font style you can easily customize

the images by changing the font size or color right in your code. This is such a useful way for creating nice looking buttons that just about every Windows Store app made by Microsoft uses this approach. So much so that there is a special button classes called **AppBarCommand** (JS) and **AppBarButton** (C#/VB). Take the following button for example.

HTML

```
<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{icon: '&#xE1C4;', type:'button'}"></button>
```

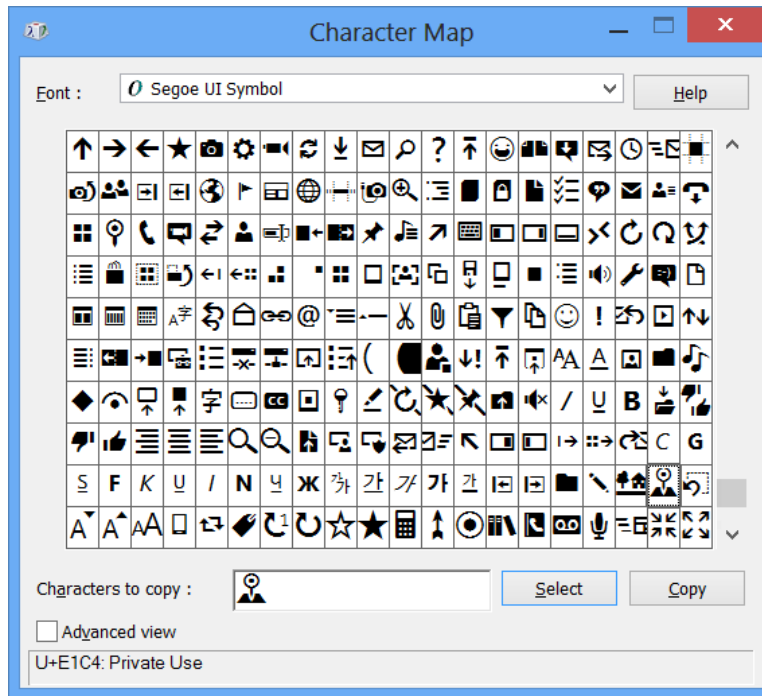
XAML

```
<AppBarButton>
  <AppBarButton.Icon>
    <FontIcon Glyph="&#xE1C4;" />
  </AppBarButton.Icon>
</AppBarButton>
```

This ends up looking like the following image which looks a lot better than the text version of the button we saw initially.



Taking a closer look at this button we see that the HTML version is creating an **AppBarCommand** and the XAML version is creating an **AppBarButton**. This defines the font family of the button as “Segoe UI Symbol” and adds a nice round border around the icon. We could have simply set the font family ourselves but then we would need to do a lot more work to get the round border. Next look at the content of the button, an icon can be specified using the **icon** property. The value of the icon looks a bit odd. The icon is actual a hex Unicode value. So how do you know what Unicode values to use? If you are using Windows 8, search for an app called “Character Map”. When this opens set the font to Segoe UI Symbol. This will show you all the possible symbols you can use.



If you hover over or double click a symbol you will see a value that looks something like U+E123. This is the Unicode value. To use it in your app simply replace U+ with &#x and a semi-colon at the end. Then add this to the content of your button.

Taking this a bit further it may be useful to also display a label under the button just in case the icon is not self-explanatory enough. To do this, simply set the **label** property of the button as shown here.

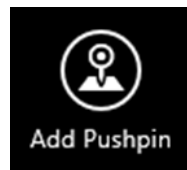
HTML

```
<button data-win-control="WinJS.UI.AppBarCommand"
  data-win-options="{icon: '&#xE1C4;', label: 'Add Pushpin', type:'button'}"></button>
```

XAML

```
<AppBarButton Label="Add Pushpin">
  <AppBarButton.Icon>
    <FontIcon Glyph="&#xE1C4;"/>
  </AppBarButton.Icon>
</AppBarButton>
```

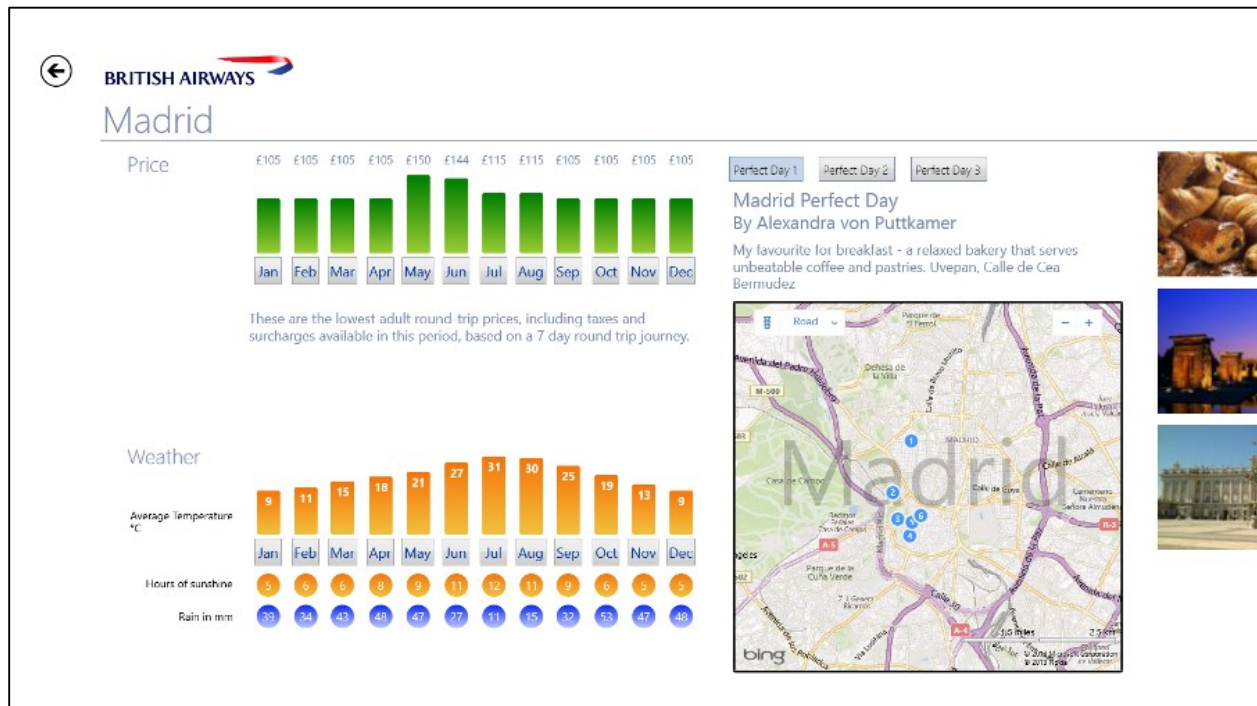
This button will end up looking like this.



Throughout the code samples in this book we will create buttons using this method.

Real World Example: The British Airways Inspiration App

British Airways has created an excellent Windows Store application called “The British Airways Inspiration App”. This app allows you to explore many of the great places that British Airways fly to, discover places that are just the right temperature, and learn how to spend a perfect day in many of the cities. This app is a great example of how a Windows Store app can be used to create a simple yet powerful user experience. The real genius behind this app is that rather than redeveloping the British Airways booking engine into a Windows Store app they took some of the great destination guide information that is often over looked on their website and turned it into an app. This allowed them to make this great information easier to find. Taking things a bit further if the user selects a month from the price or temperature tables, they will be taken to the British Airways website where they can book a flight. This is a great way to drive traffic to their existing site. You can find this app in the Windows Store here: <http://bit.ly/18vF3kB>



Chapter Summary

In this chapter you were introduced to Windows Store apps and many of the tools used when creating apps. Here is a short summary of some key points from this chapter.

- You need a developer license to create Windows Store Apps. Developer licenses are free and expire after 30 days. They can be easily renewed.
- When creating a JavaScript based app separate your JavaScript, CSS Styles and HTML in separate files. This keeps things clean and also allows your application to take advantage of byte-code caching.
- The app manifest describes your app (its name, description, tile, start page, and so on) and lists the files that your app contains. It also lists the capabilities your application has.
- The Windows Store App simulator can simulate touch events, rotate the device, change the screen size and resolution, provide location information, and capture screenshots.
- The simulator does not provide any tools for simulating sensors. If your application uses sensor you will need to test your app on a device that has the required sensors.

- Side loading allows you to install an application on another device. This requires the device to have a developer license.
- Test your apps on devices that use different types of processors. Apps that work great on an x86 or x64 processor may have performance issues on a device that uses an ARM processor.
- In Windows Store apps we have the ability to use Uri's to launch other applications that are installed on a user's computer from within our app. This is known as protocol activation and can be achieved by using the **Windows.System.Launcher** class.
- You can use protocol activation to launch the Maps app. The Map app provides a set of query parameter which can be used to load a map to a specific location.
- Bing Maps is the main mapping platform used for developing location based Windows Store Apps. The Bing Maps Windows Store SDK can be downloaded here: <http://bit.ly/11TLdei>
- The Bing Maps REST services provides you with geocoding, reverse-geocoding, routing, static imagery, elevation and traffic incident information without the need to load an interactive map.
- The Bing Spatial Data Services offer four key functionalities: batch geocoding, point of interest (POI) data, boundary data, and the ability to store and expose your spatial data.
- You can create a Bing Maps account and key at <http://bingmapsportal.com>
- A Bing Maps key is used to authenticate all requests to the Bing Maps platform.
- The maps in Bing Maps are broken up into a large number of images at different levels of detail. These images are referred to as tiles.
- Bing Maps uses a Mercator projection to stretch the spherical globe into a flat square map.
- Map tiles are identified using either a XY index or quadkey value.
- The code download that accompanies this book includes code samples of how to use the location API, all the sensors, and how to link to the Maps app. You can download the code samples from here: <http://bit.ly/1cfOtk2>
- The "Segoe UI Symbol" font family is an excellent source of graphical elements for buttons. The Character Map tool is available in Windows 8 and is a great way to look through all the available symbols.

Chapter 2: The Sensor and Location Platform

With the release of Windows 7 Microsoft added in a new location and sensors platform to Windows. The goal behind this platform was to remove the old legacy use of COM-ports for communication with external devices and providing a newer more modern development API. This platform is also in Windows 8 and will likely continue to be in future versions of Windows as well.

Overview of the Sensor API

Before we dive into all the sensors available to Windows Store apps let's first take a step back and understand what a sensor is. Sensors are hardware components that are capable of measuring the physical outside world. For example, a sensor could detect the orientation or movement of a computing device. In the Windows Store app API we have access to a number of sensors. These sensors are managed through the **Windows.Devices.Sensors** namespace. The following is a list of the different types of sensors available to Windows Store apps.

Sensor	Description
Accelerometer	Detects acceleration along three axes (x, y, and z).
Inclinometer	Detects angle of incline along three axes (pitch, roll, and yaw).
Gyrometer	Detects angular velocity along three axes.
Compass	Detects heading in degrees relative to magnetic north (and due north when integrated with a GPS).
Ambient Light Sensor	Detects ambient lighting level in lumens.
Orientation Sensor	Combines the data from the accelerometer, compass, and gyrometer sensors to provide smoother and more sensitive rotation data than can be obtained from any of the sensors alone.
Simple Orientation Sensor	Uses the accelerometer to obtain device orientation as a rotation into one of four quadrants, or face-up, or face-down.

All sensors in the **Windows.Devices.Sensors** namespace with the exception of the simple orientation sensor, share a common set of events, methods, and properties. The simple orientation sensor has the same events and methods but does not have the reporting related properties. Here are the common elements shared by all sensors (the name value includes the C#/VB name followed by the JavaScript equivalent):

Events

Name	Description
ReadingChanged readingchanged	Occurs each time the sensor reports a new reading.

Methods

Name	Description
GetCurrentReading getCurrentReading	Gets the current reading.
GetDefault getDefault	Returns a reference to the default sensor.

Properties

Name	Description
MinimumReportInterval minimumReportInterval	Gets the minimum reporting interval supported by the sensor measured in milliseconds.
ReportInterval reportInterval	Gets or sets the current reporting interval for the sensor in milliseconds.

There are two main ways in which to implement sensors. The first approach is to simply request the current reading from the sensor. Using the light sensor, here is an example of how to get the current reading.

JavaScript

```
var lightSensor = Windows.Devices.Sensors.LightSensor.getDefault();
var reading = lightSensor.getCurrentReading();
```

C#

```
LightSensor lightSensor = LightSensor.GetDefault();
LightSensorReading reading = lightSensor.GetCurrentReading();
```

Visual Basic

```
Dim lightSensor As LightSensor = lightSensor.GetDefault()
Dim reading As LightSensorReading = lightSensor.GetCurrentReading()
```

The second approach is to monitor a sensor and react to changes in measurements. This can be done by using the steps outlined as follows:

- 1) Get a reference to the default device.
- 2) Verify that the sensor exists by checking that the reference to the sensor is not null. Note that not all devices have all the sensors that are supported in Windows Store apps.
- 3) Specify an interval on which to poll the device for updates.
- 4) Add an event handler to react when to changes in the device readings.
- 5) Remove the event handler when the app shuts down.

Using the light sensor again as an example here is an example of how you can monitor changes in measurements.

JavaScript

```
function ActivateLightSensor()
{
    var lightSensor = Windows.Devices.Sensors.LightSensor.getDefault();
    if (lightSensor) {
        //Set a reasonable reporting interval for the sensor
        lightSensor.reportInterval = lightSensor.minimumReportInterval > 16 ?
            lightSensor.minimumReportInterval : 16;

        //Add a reading change event to the sensor
        lightSensor.addEventListener("readingchanged", lightSensorReadingChanged);
    } else {
        //No light sensor found. Sensor may not be supported by device.
    }
}
```

```

}

function lightSensorReadingChanged(e) {
    var reading = e.reading;
    //Do something with the readings
}

```

C#

```

private void ActivateLightSensor()
{
    LightSensor lightSensor = LightSensor.GetDefault();
    if (lightSensor != null)
    {
        //Set a reasonable reporting interval for the sensor
        uint minReportInterval = lightSensor.MinimumReportInterval;
        lightSensor.ReportInterval = minReportInterval > 16 ? minReportInterval : 16;

        //Add a reading change event to the sensor
        lightSensor.ReadingChanged += LightSensorReadingChanged;
    }
    else
    {
        //No light sensor found. Sensor may not be supported by device.
    }
}

private async void LightSensorReadingChanged(object sender,
LightSensorReadingChangedEventArgs e)
{
    try
    {
        await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            LightSensorReading reading = e.Reading;
            //Do something with the readings
        });
    }
    catch { }
}

```

Visual Basic

```

Private Sub ActivateLightSensor()
    Dim lightSensor As LightSensor = lightSensor.GetDefault()
    If lightSensor IsNot Nothing Then
        'Set a reasonable reporting interval for the sensor
        Dim minReportInterval As UInteger = lightSensor.MinimumReportInterval
        lightSensor.ReportInterval = If(minReportInterval > 16, minReportInterval, 16)

        'Add a reading change event to the sensor
        AddHandler lightSensor.ReadingChanged, AddressOf LightSensorReadingChanged
    Else
        'No light sensor found. Sensor may not be supported by device.
    End If
End Sub

```

```

Private Async Sub LightSensorReadingChanged(sender As Object, e As
LightSensorReadingChangedEventArgs)
    Try
        Await Dispatcher.RunAsync(CoreDispatcherPriority.Normal,
            Sub()
                Dim reading As LightSensorReading = e.Reading
                'Do something with the readings
            End Sub)
    Catch
    End Try
End Sub

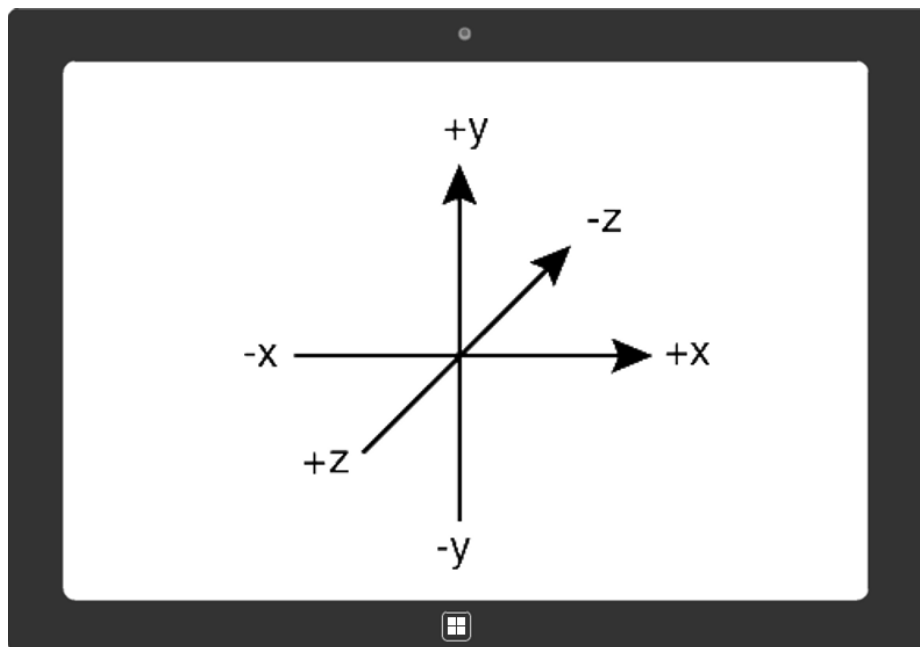
```

A couple of things to take note of managed code samples is that the event handlers are asynchronous methods. By making them asynchronous the sensor event and any processing we do with the data will be handle on a thread other than the UI thread. This will help ensure that you have a nice smooth user experience in your app. If you want to pass information back to the UI thread you will need to use a dispatcher otherwise you will get a cross thread exception.

Tip: Using **async void** should be avoided when possible and should only be used in top level event handlers, in our case a button tapped event handler. Any code inside of a method that uses **async void** should be wrapped with a try/catch block. The main reason for this is that if an exception occurs in our code it won't be able to bubble up and out of the method. This will cause the application to crash.

Tip: It is a good practice to remove any event handlers to sensors when the user control in which they are providing data to becomes no longer visible. This will free up the resources that are being used by the sensor. You can then re-add the event handlers when the user control is made visible again.

Each sensor has unique class that is used to store the reading information. Many of these readings that are relative to the x, y and z axis of the device. When facing the screen of the device the x-axis runs in a left to right direction, the y-axis runs in a bottom to top direction, and the z-axis runs from the back of the screen through to the front. The following diagram demonstrates how these axis relate to a device.



Important Note: At the time of writing this book the Windows Store Simulator does not provide any tools for simulating sensor readings. If you are writing an app that uses sensors be sure to test it on a device that has those sensors. This is where side loading your app onto another device can be useful.

Accelerometer

An accelerometer gives your app the ability to measure acceleration forces. These forces may be static, like the constant force of gravity pulling at your feet, or they could be dynamic which might be caused by moving or vibrating the accelerometer. The accelerometer itself measures the force of gravity, and the motion of the device itself and tracks the acceleration along 3 axes (x, y, and z). The values returned by this sensor are measured in g-forces where 1g is equivalent to 9.81m/s^2 .

Take for instance a tablet that is lying flat on its back, the z-axis would align perfectly with the force of gravity. This will result in an acceleration in the z-axis of -1g as the force of gravity is being applied through the front of the screen which is the direction of negative z-axis. While in this position, if the device is sitting still the x and y values of the accelerometer should be close to zero. Turning the device on its edge will cause either the x or y values to be + or - 1g.

Accelerometers have a number of useful purposes in computing devices such as being able to determine which way is up so as to rotate the screen accordingly. This type of sensor is also often used by hard drives to detect when a device is falling or shaken so that it can lock the hard drive read arm away so as to prevent scratching the disk platters. In the world of gaming the accelerometer can be used to turn the device into a controller such that when the user turns the device a movement occurs in the app. From a location intelligence point of view an accelerometer can be used to determine changes in direction and speed.

When the **ReadingChanged** event for the accelerometer fires the event handler receives an **AccelerometerReadingChangedEventArgs** object class. This class has one property that contains the reading information as an **AccelerometerReading** class and has the following properties.

Name	Description
AccelerationX accelerationX	Gets the g-force acceleration along the x-axis.
AccelerationY accelerationY	Gets the g-force acceleration along the y-axis.
AccelerationZ accelerationZ	Gets the g-force acceleration along the z-axis.
Timestamp timestamp	Gets the time at which the sensor reported the reading.

In addition to the standard **ReadingChanged** event used by sensors, the accelerometer also exposes a second event called **Shaken**. The Shaken event fires when the device has been shaken for a short period of time. This event doesn't return any useful information but does allow use to react to this type of event. The following

Tip: If you are creating a cross platform application for Windows Store and Windows Phone 8 it should be noted that the accelerometer sensor does not raise the Shaken event on the Windows Phone 8 platform. If you add an event handler for the Shaken event in your Windows Phone 8 app, no error is raised, but the code in the event handler won't run.

Compass

A compass sensor returns a heading with respect to True North. On some devices it may also return a value for Magnetic North if the sensor on the device supports it. When the **ReadingChanged** event for the compass fires the event handler receives a **CompassReadingChangedEventArgs** object class. This class has one property that contains the reading information as a **CompassReading** class and has the following properties.

Name	Description
HeadingMagneticNorth headingMagneticNorth	Gets the heading in degrees relative to magnetic-north.

HeadingTrueNorth headingTrueNorth	Gets the heading in degrees relative to geographic true-north.
Timestamp timestamp	Gets the time at which the sensor reported the reading.

Gyrometer

A gyrometer sensor measures angular velocity (speed) of a device when it is moved. These measurements are taken along the x, y, and z axis. When the device is not moving these values will be zero. The angular velocity is measured in degrees per second.

When the **ReadingChanged** event for the gyrometer fires the event handler receives a **GyrometerReadingChangedEventArgs** object class. This class has one property that contains the reading information as a **GyrometerReading** class and has the following properties.

Name	Description
AngularVelocityX angularVelocityX	Gets the angular velocity about the x-axis.
AngularVelocityY angularVelocityY	Gets the angular velocity about the y-axis.
AngularVelocityZ angularVelocityZ	Gets the angular velocity about the z-axis.
Timestamp timestamp	Gets the time at which the sensor reported the reading.

Inclinometer

The inclinometer measures the angle of rotation around the axis and exposes these angles as yaw, pitch, and roll readings. The yaw is the z rotation, the roll is the y rotation, and the pitch is the x rotation of the device. Inclinometers are often used to measure relative to horizon. By combining an inclinometer with some basic trigonometry you can use it to measure the heights of tall objects. Simply go to the base of the object, walk a known distance away and then tilt the device until it points to the top of the object. Measure the angle and use the following formula to determine the height.

$$\text{Height} = \text{Distance from base} * \tan(\text{angle to horizon})$$

When the **ReadingChanged** event for the inclinometer fires the event handler receives an **InclinometerReadingChangedEventArgs** object class. This class has one property that contains the reading information as an **InclinometerReading** class and has the following properties.

Name	Description
PitchDegrees pitchDegrees	Gets the rotation in degrees around the x-axis.
RollDegrees rollDegrees	Gets the rotation in degrees around the y-axis.
YawDegrees yawDegrees	Gets the rotation in degrees around the z-axis.
Timestamp timestamp	Gets the time at which the sensor reported the reading.

Light Sensor

A light sensor is a device that is used to detect light. It does this by measuring the ambient light and returns brightness information as a lux value. A lux is a standard unit of measurement that represents the total amount of visible light in a square meter. This type of sensor can be used for various purposes, the most common of which is to adjust the brightness of a screen as the light changes in a room.

When the **ReadingChanged** event for the light sensor fires the event handler receives a **LightSensorReadingChangedEventArgs** object class. This class has one property that contains the reading information as a **LightSensorReading** class and has the following properties.

Name	Description
IlluminanceInLux illuminanceInLux	Gets the luminance level in lux.
Timestamp timestamp	Gets the time at which the sensor reported the reading.

OrientationSensor

The OrientationSensor combines the accelerometer, compass, and gyrometer to report even more sensitive movement than the sensors can on their own. This sensor is often used in more complex apps that use the device to control the app. This makes the movement within the app much smoother but also makes the mathematics much more complicated.

Another location intelligence related use case is to combine this sensor with the location API and a camera to create an app that not only can display where a picture was taken but also show the direction in which it was taken. This information could then be used to show the location of what you were taking a picture of rather than where you were when you took the picture.

When the **ReadingChanged** event for the orientation sensor fires the event handler receives an **OrientationSensorReadingChangedEventArgs** object class. This class has one property that contains the reading information as an **OrientationSensorReading** class and has the following properties.

Name	Description
Quaternion quaternion	Gets the quaternion for the current orientation-sensor reading.
RotationMatrix rotationMatrix	Gets the rotation matrix for the current orientation-sensor reading.
Timestamp timestamp	Gets the time at which the sensor reported the reading.

A quaternion represents two things, an axis on which a rotation occurs and the rotation around that axis. It has an x, y, and z component, which represents the axis on which the rotation occurs. It also has a w component, which represents the amount of rotation that occurs around this axis. The quaternion has the following properties.

Name	Description
W w	Gets the w-value of the quaternion.
X x	Gets the x-value of the quaternion.
Y y	Gets the y-value of the quaternion.
Z z	Gets the z-value of the quaternion.
Timestamp timestamp	Gets the time at which the sensor reported the reading.

A rotation matrix is a matrix that is used to perform a rotation in the Cartesian coordinate system. Since the orientation sensor measures the orientation of a device in 3 dimensions the rotation matrix is a 3x3 matrix. The rotation matrix has the following properties.

Name	Description
M11 m11	Gets the value at row 1, column 1 of the given rotation matrix.
M12 m12	Gets the value at row 1, column 2 of the given rotation matrix.
M13 m13	Gets the value at row 1, column 3 of the given rotation matrix.
M21 m21	Gets the value at row 2, column 1 of the given rotation matrix.
M22 m22	Gets the value at row 2, column 2 of the given rotation matrix.
M23 m23	Gets the value at row 2, column 3 of the given rotation matrix.
M31 m31	Gets the value at row 3, column 1 of the given rotation matrix.
M32 m32	Gets the value at row 3, column 2 of the given rotation matrix.
M33 m33	Gets the value at row 3, column 3 of the given rotation matrix.
Timestamp timestamp	Gets the time at which the sensor reported the reading.

SimpleOrientationSensor

This sensor detects the current quadrant orientation of the specified device as well as its face-up or face-down status. This sensor simplifies the orientation sensor by converting the numbers into standard orientation states and returning them as a human readable value. This sensor is often used to detect when the user has change the orientation of the screen.

When the **ReadingChanged** event for the simple orientation sensor fires the event handler receives a **SimpleOrientationSensorOrientationChangedEventArgs** object class. This class has one property called **Orientation** which is a **SimpleOrientation** enumerator. This following is a list of the different values returned by this enumerator. Note that the name of the enumerator member will start with a capital if being used in C# or Visual Basic, and start with a small letter if being used in JavaScript.

Member Name	Value	Description
NotRotated notRotated	0	The device is not rotated. This corresponds to portrait-up.
Rotated90DegreesCounterclockwise rotated90DegreesCounterclockwise	1	The device is rotated 90-degrees counter-clockwise. This corresponds to landscape-left.
Rotated180DegreesCounterclockwise rotated180DegreesCounterclockwise	2	The device is rotated 180-degrees counter-clockwise. This corresponds to portrait-down.
Rotated270DegreesCounterclockwise rotated270DegreesCounterclockwise	3	The device is rotated 270-degrees counter-clockwise. This corresponds to landscape-right.
Faceup faceup	4	The device is face-up and the display is visible to the user.
Facedown facedown	5	The device is face-down and the display is hidden from the user.

The Location Platform

The Location API is built on to top of the sensor platform and generates geographic data for apps. One of the key new features in this platform is the ability for multiple applications to access the same sensor at the same time. In the past this was an issue with GPS devices where only one application could access the GPS coordinates at a time. There are number of different methods for determining the location of a device, each having a different level of accuracy.

Locating method	Accuracy
Global Positioning System (GPS)	5 meters - 15 meters
Wi-Fi triangulation	10 meters - 350 meters
Cell phone tower triangulation	40 meters - 5 kilometers
IP address resolution	Within 50 kilometers

The Location API determines the most accurate location sensor available on a device. This simplifies programming because the Location API will only report the data from the most accurate available sensor, even when there are multiple location sensors available. Windows Store apps can access the location API through the **Windows.Devices.Geolocation** namespace. Most Windows Store apps will have access to Wi-Fi triangulation and IP address data. GPS and Cell phone tower triangulation methods are not yet widely integrated into standard computing devices like they are in mobile devices. However, if a GPS device were connected and providing data the Location API will use the sensor automatically.

The main class in the **Windows.Devices.Geolocation** namespace for getting a user's location is the **Geolocator** class. This class has the following events, methods and properties.

Events

Name	Description
------	-------------

PositionChanged positionchanged	Raised when the location is updated.
StatusChanged statuschanged	Raised when the ability of the Geolocator to provide updated location changes.

Methods

Name	Description
GetGeopositionAsync getGeopositionAsync	Starts an asynchronous operation to retrieve the location of the user's computer.

Properties

Name	Description
DesiredAccuracy desiredAccuracy	The accuracy level at which the Geolocator provides location updates.
LocationStatus locationStatus	The status that indicates the ability of the Geolocator to provide location updates.
MovementThreshold movementThreshold	Gets and sets the distance of movement, in meters, relative to the coordinate from the last PositionChanged event that is required for the Geolocator to raise a PositionChanged event.
ReportInterval reportInterval	The requested minimum time interval between location updates, in milliseconds.

Tip: If your app requires updates infrequently, use the movement threshold and report interval values so that the location provider can conserve power by calculating locations only when needed. Many real-time tracking applications typically use an update frequency of 1 to 10 minutes depending on the use case. There are only a small number of apps that need to a user's location more frequently.

Similar to how the other sensors work there are two main ways in which to implement the **Geolocator** class. The first approach is to simply request the current location using the **GetPositionAsync** method. Note that this is an **async** method so we have to use the **await** keyword when calling this method and must also wrap this inside of an **async** method.

JavaScript

```
var geolocator = new Windows.Devices.Geolocation.Geolocator();

geolocator.getGeopositionAsync().then(
    function (location) {
        //Do something with the location
    },
    function (err) {
        //Error getting position
    }
);
```

C#

```
private async void GetCurrentLocation()
{
    try
    {
        Geolocator geolocator = new Geolocator();
```



```

        Geoposition location = await geolocator.GetGeopositionAsync();
        //Do something with the location
    }
    catch { }
}

```

Visual Basic

```

Private Async Sub GetCurrentLocation()
    Try
        Dim geolocator As Geolocator = New Geolocator()
        Dim location As Geoposition = Await geolocator.GetGeopositionAsync()
        'Do something with the location
    Catch
    End Try
End Sub

```

The second approach is to monitor the position of the device and react when it changes. This can be done by using the steps outlined as follows:

- 1) Create a new instance of the **Geolocator** class.
- 2) Optionally set a movement threshold and/or a report interval.
- 3) Add an event handler(s) to react when the position and/or status change.
- 4) Remove the event handler(s) when the app shuts down.

Using the light sensor again as an example here is an example of how you can monitor changes in measurements.

JavaScript

```

function ActivateGeolocator() {
    var geolocator = new Windows.Devices.Geolocation.Geolocator();
    geolocator.addEventListener("positionchanged", positionChanged);
    geolocator.addEventListener("statuschanged", statusChanged);
}

function positionChanged(e){
    var location = e.position;
    //Do something with the location
}

function statusChanged(e) {
    var status = e.status;
    //Do something with the status
}

```

C#

```

private void ActivateGeolocator()
{
    Geolocator geolocator = new Geolocator();
    geolocator.PositionChanged += new PositionChanged;
    geolocator.StatusChanged += new StatusChanged;
}

private async void PositionChanged(object sender, PositionChangedEventArgs e)
{
    try

```

```

    {
        await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            Geoposition location = e.Position;
            //Do something with the location
        });
    }
    catch { }
}

private void StatusChanged(object sender, StatusChangedEventArgs e)
{
    PositionStatus status = e.Status;
    //Do something with the status
}

```

Visual Basic

```

Private Sub ActivateGeolocator()
    Dim geolocator As Geolocator = New Geolocator()
    AddHandler geolocator.PositionChanged, AddressOf PositionChanged
    AddHandler geolocator.StatusChanged, AddressOf StatusChanged
End Sub

Private Async Sub PositionChanged(sender As Object, e As PositionChangedEventArgs)
    Try
        Await Dispatcher.RunAsync(CoreDispatcherPriority.Normal,
            Sub()
                Dim location As Geoposition = e.Position
                'Do something with the location
            End Sub)
    Catch
    End Try
End Sub

Private Async Sub StatusChanged(sender As Object, e As StatusChangedEventArgs)
    Dim status As PositionStatus = e.Status
    'Do something with the status
End Sub

```

When the **PositionChanged** event fires, the event handler receives a **PositionChangedEventArgs** object class. This class has one property called **Position** which is a **Geoposition** class and has the following properties.

Name	Description
CivicAddress civicAddress	A CivicAddress object that contains address data associated with a geographic location.
Coordinate coordinate	A Geocoordinate object that contains the latitude and longitude associated with a geographic location.

Both of these properties are complex objects. The **CivicAddress** class contains address information for a location and has the following properties.

Name	Description
City city	The name of the city.
Country country	The name of the country, represented by using a two-letter ISO-3166 country code. A complete list of ISO-3166 two letter country codes is available in Appendix A.

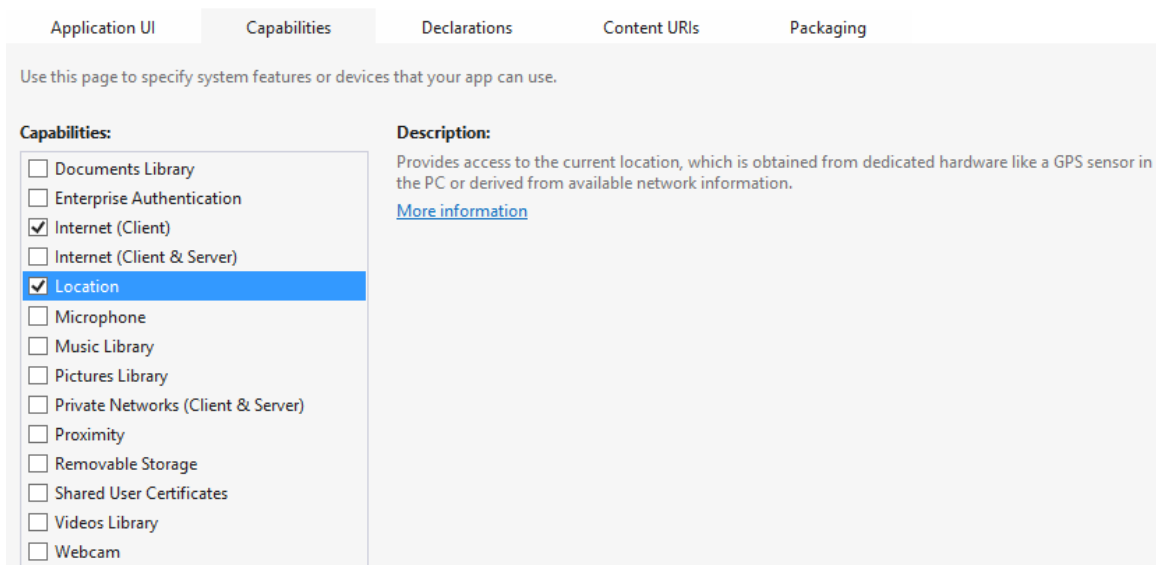
PostalCode postalCode	The postal code of the current location.
State state	The name of the state or province.
Timestamp timestamp	The time at which the location data was obtained.

The **Geocoordinate** class contains the location information and has the following properties.

Name	Description
Accuracy accuracy	The accuracy of the location in meters.
AltitudeAccuracy altitudeAccuracy	The accuracy of the altitude, in meters.
Heading heading	The current heading in degrees relative to true north.
Point point	Contains the latitude and longitude coordinate information.
Speed speed	The speed in meters per second.
Timestamp timestamp	The time at which the location was determined.

Values for the **Point** and **Accuracy** properties are always provided. Values for the **Altitude**, **AltitudeAccuracy**, **Heading**, and **Speed** properties are provided if available. If they are not available, they will be null in JavaScript code.

Before your application can retrieve a user's location you need to add the Location platform as a capability of your app in the package manifest. To do this, open the **package.appxmanifest** file and select the Capabilities tab at the top. Under the Capabilities section check the Location option.



If your application is having issues retrieving a user's location, check the following:

- Ensure that you have enabled access to the location platform in your app by opening the app manifest and checking the Location checkbox in the Capabilities tab.
- Verify that the location services are enabled on the device. To do this go to the Control Panel, open Location Settings and confirm that "Turn on the Windows Location platform" option is checked.
- Ensure you have Wi-Fi enabled on your device. If you don't then your device will be unable to use Wi-Fi triangulation or do IP address lookups.

Alternate Location Providers

If you are developing using JavaScript and HTML you have a couple of other options for accessing the user's location. Since these types of applications are rendered using the built in browser in Windows you have access to the W3C Geolocation API (<http://bit.ly/10lbUwd>) which is a standard feature in most modern browsers.

Another option, if you are using the Bing Maps SDK in JavaScript is to make use of the **GeoLocationProvider** class which wraps and simplifies the W3C Geolocation API. In addition to this it also generates a circle of accuracy which can be displayed on the map for you which is a nice feature. We will discuss the **GeoLocationProvider** more in Chapter 3. Both of these options are good choices if you are creating a cross platform application. Note that you will still need to add the location option to your app capabilities in the app manifest.

Geofencing API

Geofencing is a new feature in Windows 8.1 that allows an app to define a geographical region which can be used to trigger events when the device running the app enters or exists the region. The following is a list of some things that can be done with the geofencing API:

- Create one or more geofences, or areas of interest.
- Request to be notified when the device enters or leaves a geofence.
- Specify a time window during which the geofence is active.
- Specify the amount of time that the device should be in or out of the geofence before the notification is triggered (dwell time).
- Receive geofence events while the app is active.
- Register with the system to have a background task launch when the state of one of your app's geofences changes. Only apps that are on the lock screen can be launched in the background. For more information, see Lock Screen Overview.

There are a number of different scenarios where geofencing can be used. For instance, notifying the user with a reminder they are leaving work or home. If creating a business related app, geofences could be used to notify users about a promotion when they are near a store. Taking geofences beyond a single user experience you could send notifications to a cloud service like the Windows Azure Mobile services and notify others when a device enters or leaves an area. For example, this could be used to create an app that automatically checks you into places when you arrive without needing the user to press a button.

The **Windows.Devices.Geolocation.Geofencing** namespace contains the classes needed to do geofencing. To make use of the geofencing functionality in your app you first need to create one or more **Geofence** objects that define the areas of interest and the conditions for notification. Once that is done you must then setup event handlers for when the device entering or exiting one of the geofences. All the properties of the **Geofence** class are read only. As such all options should be set when creating an instance of the **Geofence** class. The following is what the constructor looks like for the **Geofence** class.

```
Geofence(id, geoshape, monitoredStates, singleUse, dwellTime, startTime, duration)
```

The following is a description of all the parameters that can be used to create an instance of the **Geofence** class.

Name	Type	Description
id	string	The Id of the geofence.
geoshape	IGeoshape	The area that defines the geofence to monitor.
monitoredStates	MonitoredGeofenceStates	The states to monitor the geofence for. Optional, default to monitor for both the Entered and Exited states.
singleUse	bool	True indicates the geofence should be monitored only for one use. False indicates the geofence should be monitored for multiple uses. Optional, defaults to false.
dwellTime	number (JS) TimeSpan (C#/VB)	The time that a position has to be in or out of the geofence in order for the notification to be triggered. Optional, defaults to 10 seconds.

startTime	Date (JS) DateTimeOffset	The time to start monitoring the geofence. Optional, defaults to 0 seconds which means it will start immediately.
duration	number (JS) TimeSpan (C#/VB)	The duration of time to monitor the geofence for. The duration begins at the startTime . Optional, defaults to 0 when means it will run forever.

The **IGeoshape** interface is inherited by classes that be used to define a spatial area and determine if a coordinate is within the area or not. There are two classes that are in that API that inherit from this class, the **Geocircle** and **Geopoint** classes. The **Geocircle** class will likely be the class most apps will make use of. All the properties of the **Geocircle** class are read only. As such all options should be set when creating an instance of the **Geocircle** class. The following is what the constructor looks like for the **Geocircle** class.

```
Geocircle(position, radius, altitudeReferenceSystem, spatialReferenceld)
```

The following is a description of all the parameters that can be used to create an instance of the **Geocircle** class.

Name	Type	Description
position	BasicGeoposition	The geographic center of the circle.
radius	number	The radius of the circle in meters. The valid range of radius values is from .1 to 10018754.3 meters. 10018754.3 meters is one quarter of the earth's circumference.
altitudeReferenceSystem	AltitudeReferenceSystem	The altitude reference system of the new circle. Optional.
spatialReferenceld	number	The spatial reference identifier for the geographic circle, corresponding to a spatial reference system based on the specific ellipsoid used for either flat-earth mapping or round-earth mapping. Optional, defaults to 4326.

The spatial reference id (SRID) corresponds to a spatial reference system based on the specific ellipsoid used for either flat-earth mapping or round-earth mapping. Spatial instances with the same SRID can be used when performing operations with spatial data methods on your data. The result of any spatial method derived from two spatial data instances is valid only if those instances have the same SRID that is based on the same unit of measurement, datum, and projection used to determine the coordinates of the instances. The most common units of measurement of a SRID are meters or square meters. The default SRID for the Windows platform is 4326 which is WGS84 ellipsoid.

The spatial reference identification system is defined by the European Petroleum Survey Group (EPSG) standard, which is a set of standards developed for cartography, surveying, and geodetic data storage. This standard is owned by the Oil and Gas Producers (OGP) Surveying and Positioning Committee.

The **BasicGeoposition** struct is a simple object that has the following properties.

Name	Type	Description
Altitude altitude	number	The altitude of the geographic position.
Latitude latitude	number	The latitude of the geographic position. The valid range of latitude values is from -90.0 to 90.0 degrees.
Longitude longitude	number	The longitude of the geographic position. This can be any value. For values less than or equal to -180.0 or values greater than 180.0, the value may be wrapped and stored appropriately before it is used. For

		example, a longitude of 183.0 degrees would become -177.0 degrees.
--	--	--

MonitoredGeofenceStates Enumerator

This enumerator indicates the state or states of the Geofences that are currently being monitored by the system.

Name	Value	Description
None none	0	No flag is set.
Entered entered	1	The device has entered a geofence area.
Exited exited	2	The device has left a geofence area.
Removed removed	4	The geofence has been removed.

AltitudeReferenceSystem Enumerator

This enumerator indicates the altitude reference system to be used in defining a geographic shape.

Name	Value	Description
Unspecified unspecified	0	The altitude reference system was not specified.
Terrain terrain	1	The altitude reference system is based on distance above terrain or ground level.
Ellipsoid ellipsoid	2	The altitude reference system is based on an ellipsoid which is a mathematical approximation of the shape of the Earth.
Geoid geoid	3	The altitude reference system is based on the distance above sea level.
Surface surface	4	The altitude reference system is based on the distance above the tallest surface structures, such as buildings, trees, roads, etc., above terrain or ground level.

Once a **Geofence** object is created it can be added to the **Current.Geofences** property of the **GeofenceMonitor** class that is available inside apps. The **GeofenceMonitor** class is responsible maintaining information about the geofences and trigger the events on them as the device enters or leaves the geofences. The **GeofenceMonitor** class has the following elements:

Events

Name	Description
GeofenceStateChanged geofencestatechanged	Raised when the state of one or more Geofence objects in the Geofences collection of the GeofenceMonitor has changed
StatusChanged statuschanged	Raised when the state of the GeofenceMonitor has changed.

Methods

Name	Description
ReadReports readReports	Gets a collection of status changes to the Geofence objects in the Geofences collection of the GeofenceMonitor .

Properties

Name	Description
Current current	Gets the GeofenceMonitor object which contains all of an app's Geofence information.
Geofences geofences	Returns a reference to the collection Geofence objects currently registered with the system wide GeofenceMonitor .

LastKnownGeoposition lastKnownGeoposition	Last reading of the device's location.
Status status	Indicates the current state of the GeofenceMonitor .

The following methods show how geofences can be added and removed from your application.

JavaScript

```
function addGeofence(id, latitude, longitude, radius) {
    //Remove any geofences that might already exist with this id.
    removeGeofence(id);

    //Define the center of the Geofence.
    //Use a JavaScript object that has a latitude, longitude and altitude value to create a
    BasicGeopoint object
    var position = {
        latitude: latitude,
        longitude: longitude,
        altitude: 0
    };

    //Create a Geocircle from the center location and the radius.
    var geocircle = new Windows.Devices.Geolocation.Geocircle(position, radius);

    //Define which states to monitor
    var monitoredStates =
        Windows.Devices.Geolocation.Geofencing.MonitoredGeofenceStates.entered |
        Windows.Devices.Geolocation.Geofencing.MonitoredGeofenceStates.exited |
        Windows.Devices.Geolocation.Geofencing.MonitoredGeofenceStates.removed;

    //Create a Geofence
    var geofence = new Windows.Devices.Geolocation.Geofencing.Geofence(id, geocircle,
        monitoredStates, false, 3);

    //Add the geofence to the GeofenceMonitor
    Windows.Devices.Geolocation.Geofencing.GeofenceMonitor.current.geofences.push(geofence);
}

function removeGeofence(id)
{
    var fences = Windows.Devices.Geolocation.Geofencing.GeofenceMonitor.current.geofences;
    //Loop through all the geofences and remove any that have the matching id.
    for (var i = 0; i < fences.length; i++)
    {
        if (fences[i].id == id)
        {
            Windows.Devices.Geolocation.Geofencing.GeofenceMonitor.current.geofences.removeAt(i);
            break;
        }
    }
}
```

C#

```
private void AddGeofence(string id, double latitude, double longitude, double radius)
{
    //Remove any geofences that might already exist with this id.
```

```

RemoveGeofence(id);

//Define the center of the Geofence.
var position = new BasicGeoposition
{
    Latitude = latitude,
    Longitude = longitude
};

//Create a Geocircle from the center location and the radius.
var geocircle = new Geocircle(position, radius);

//Define which states to monitor
var monitoredStates = MonitoredGeofenceStates.Entered | MonitoredGeofenceStates.Exited |
MonitoredGeofenceStates.Removed;

//Define a dwell time of 3 seconds before firing the event.
var dwellTime = new TimeSpan(0,0,3);

//Create a Geofence
var geofence = new Geofence(id, geocircle, monitoredStates, false, dwellTime);

//Add the geofence to the GeofenceMonitor
GeofenceMonitor.Current.Geofences.Add(geofence);
}

private void RemoveGeofence(string id)
{
    //Loop through all the geofences and remove any that have the matching id.
    foreach (var gf in GeofenceMonitor.Current.Geofences)
    {
        if (string.Compare(gf.Id, id) == 0)
        {
            GeofenceMonitor.Current.Geofences.Remove(gf);
            break;
        }
    }
}

```

Visual Basic

```

Private Sub AddGeofence(id As String, latitude As Double, longitude As Double, radius As
Double)
    'Remove any geofences that might already exist with this id.
    RemoveGeofence(id)

    'Define the center of the Geofence.
    Dim position = New BasicGeoposition()
    position.Latitude = latitude
    position.Longitude = longitude

    'Create a Geocircle from the center location and the radius.
    Dim geocircle = New Geocircle(position, radius)

    'Define which states to monitor
    Dim monitoredStates = MonitoredGeofenceStates.Entered Or MonitoredGeofenceStates.Exited
Or MonitoredGeofenceStates.Removed

    'Define a dwell time of 3 seconds before firing the event.
    Dim dwellTime = New System.TimeSpan(0, 0, 3)

```



```

'Create a Geofence
Dim geofence = New Geofence(id, geocircle, monitoredStates, False, dwellTime)

'Add the geofence to the GeofenceMonitor
GeofenceMonitor.Current.Geofences.Add(geofence)
End Sub

Private Sub RemoveGeofence(id As String)
'Loop through all the geofences and remove any that have the matching id.
For Each gf As Geofence In GeofenceMonitor.Current.Geofences
    If String.Compare(gf.Id, id) = 0 Then
        GeofenceMonitor.Current.Geofences.Remove(gf)
        Exit For
    End If
Next
End Sub

```

To implement these methods call the **AddGeofence** to add a geofence to be monitored. Attach an event handler to the **GeofenceStateChanged** event. This event will be triggered whenever the device enters or leaves the geofence. If you want to geofence to be enabled when the application starts do this in the **OnNavigatedTo** or **onloaded** event handler. Also, remember to remove the **GeofenceStateChanged** event when the **OnNavigatingFrom** or **onunload** event is fired. The following code demonstrates how to do this.

JavaScript

```

var app = WinJS.Application;

app.onloaded = function (args) {
    //Add a geofence for the center of the globe with a radius of 100 meters.
    addGeofence(geofenceId, 0, 0, 100);

    //Add an event handler for when the Geofence state changes
    Windows.Devices.Geolocation.Geofencing.GeofenceMonitor.current.addEventListener('geofencestatechanged', onGeofenceStateChanged);
};

app.onunload = function (args) {
    //Remove the event handler for when the Geofence state changes
    Windows.Devices.Geolocation.Geofencing.GeofenceMonitor.current.removeEventListener('geofencestatechanged', onGeofenceStateChanged);
};

function onGeofenceStateChanged(event)
{
    //Read the reports from the geofence monitor
    var reports = event.target.readReports();

    //Loop through the reports and display the state of the Geofence
    for(var i = 0; i < reports.length; i++)
    {
        var msg = '';

        switch(reports[i].newState)
        {
            case Windows.Devices.Geolocation.Geofencing.GeofenceState.entered:
                msg = 'Entered geofence - ' + reports[i].geofence.id;
                break;
            case Windows.Devices.Geolocation.Geofencing.GeofenceState.exited:

```

```

        msg = 'Exited geofence - ' + reports[i].geofence.id;
        break;
    case Windows.Devices.Geolocation.Geofencing.GeofenceState.removed:
        msg = 'Removed geofence - ' + reports[i].geofence.id;
        break;
    case Windows.Devices.Geolocation.Geofencing.GeofenceState.none:
    default:
        break;
    }

    document.getElementById("GeofenceData").innerHTML = msg;
}
}

```

C#

```

protected override void OnNavigatedTo(NavigationEventArgs e)
{
    //Add a geofence for the center of the globe with a radius of 100 meters.
    AddGeofence(geofenceId, 0, 0, 100);

    //Add an event handler for when the Geofence state changes
    GeofenceMonitor.Current.GeofenceStateChanged += OnGeofenceStateChanged;
}

protected override void OnNavigatingFrom(NavigatingCancelEventArgs e)
{
    //Remove the event handler for when the Geofence state changes
    GeofenceMonitor.Current.GeofenceStateChanged -= OnGeofenceStateChanged;
}

private async void OnGeofenceStateChanged(GeofenceMonitor sender, object e)
{
    try
    {
        //Read the reports from the geofence monitor
        var reports = sender.ReadReports();

        await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
        {
            //Loop through the reports and display the state of the Geofence
            foreach (var report in reports)
            {
                string msg = string.Empty;

                switch (report.NewState)
                {
                    case GeofenceState.Entered:
                        msg = string.Format("Entered geofence - {0}", report.Geofence.Id);
                        break;
                    case GeofenceState.Exited:
                        msg = string.Format("Exited geofence - {0}", report.Geofence.Id);
                        break;
                    case GeofenceState.Removed:
                        msg = string.Format("Removed geofence - {0}", report.Geofence.Id);
                        break;
                    case GeofenceState.None:
                    default:
                        break;
                }
            }
        });
    }
}

```

```

        GeofenceData.Text = msg;
    }
    });
}
catch { }
}

```

Visual Basic

```

Protected Overrides Sub OnNavigatedTo(e As Navigation.NavigationEventArgs)
    'Add a geofence for the center of the globe with a radius of 100 meters.
    AddGeofence(geofenceId, 0, 0, 100)

    'Add an event handler for when the Geofence state changes
    AddHandler GeofenceMonitor.Current.GeofenceStateChanged, AddressOf
OnGeofenceStateChanged
End Sub

Protected Overrides Sub OnNavigatingFrom(e As NavigatingCancelEventArgs)
    'Remove the event handler for when the Geofence state changes
    RemoveHandler GeofenceMonitor.Current.GeofenceStateChanged, AddressOf
OnGeofenceStateChanged
End Sub

Private Async Sub OnGeofenceStateChanged(sender As GeofenceMonitor, e As Object)
    Try
        'Read the reports from the geofence monitor
        Dim reports = sender.ReadReports()

        Await Dispatcher.RunAsync(CoreDispatcherPriority.Normal,
            Sub()
                'Loop through the reports and display the state of the Geofence
                For Each report As GeofenceStateChangeReport In reports
                    Dim msg = String.Empty

                    Select Case report.NewState
                        Case GeofenceState.Entered
                            msg = String.Format("Entered geofence - {0}",
report.Geofence.Id)

                            Exit Select
                        Case GeofenceState.Exited
                            msg = String.Format("Exited geofence - {0}",
report.Geofence.Id)

                            Exit Select
                        Case GeofenceState.Removed
                            msg = String.Format("Removed geofence - {0}",
report.Geofence.Id)

                            Exit Select
                        Case GeofenceState.None
                        Case Else
                            Exit Select
                    End Select

                    GeofenceData.Text = msg

                Next
            End Sub)

    Catch
    End Try
End Sub

```

Tip: If you want your app to be launched in the background when the trigger condition for one of its geofences is met, you need to use a background task and set up a **LocationTrigger** to launch it. You can find documentation on how to do this here: <http://bit.ly/199c5Ff>

Testing Geofences

Testing and debugging geofencing apps can be challenging because they depend on a device's location. There are two main ways to test geofences.

- Physically move the device to new locations. This may be an option if you are working with small geofences and your device is able to determine its location when both inside and out of the geofence.
- An alternative method which is likely to be easier is to use the Visual Studio simulator to simulate locations for the device.

To test your geofencing app running the foreground:

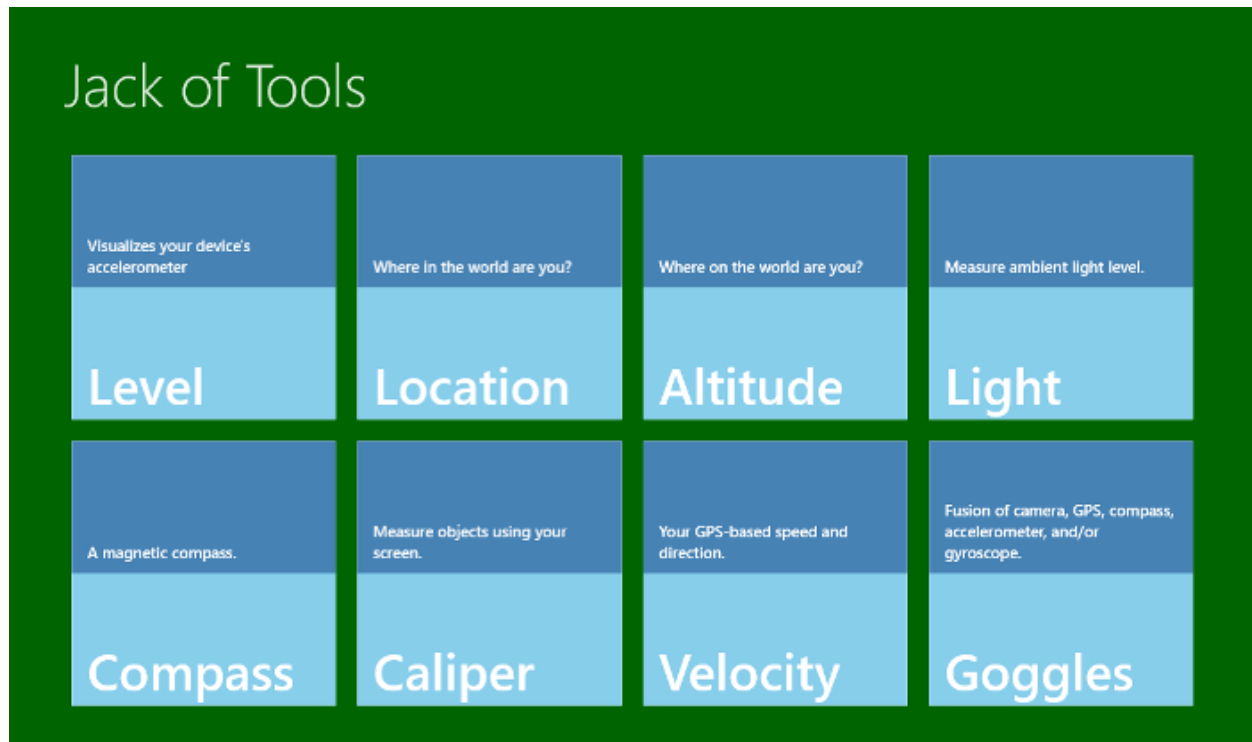
- Build your app in Visual Studio.
- Launch your app in the Visual Studio simulator.
- From the simulator, simulate various locations inside and outside of your geofence region. Be sure to wait long enough past the time specified by the **DwellTime** property to trigger the event. Note that you must accept the prompt to enable location permissions for the app.

To test your geofencing app running the background:

- Build your app in Visual Studio. Note that your app should set the Location background task type and set the badge or badge and tile assets for the lock screen.
- Deploy the app locally first and add the app to the lock screen. You can do this by accepting the prompt to add the app to the lock screen or by manually adding the app to the lock screen in Settings.
- Close your app that is running locally.
- Launch your app in the Visual Studio simulator. Note that background geofencing simulation is supported on only one app at a time within the simulator. Do not launch multiple geofencing apps within the simulator.
- From the simulator, simulate various locations inside and outside of your geofence region. Be sure to wait long enough past the **DwellTime** to trigger the event. Note that you must accept the prompt to enable location permissions for the app.
- Use Visual Studio to trigger the location background task. Information on how to trigger background tasks can be found here: <http://bit.ly/1b1b0TR>

Real World Example: Jack of Tools

Jack of Tools is a must have app if you are working with any of the sensors in your Windows Store app. This app visualizes the device sensors into useful and practical tools like a level, compass, speedometer with heading, altimeter, caliper, light meter and more. The Level for example puts your accelerometer to work with fluidly animating 2D and 1D floating levels. The Goggles tool allows you to track location, direction, orientation, and speed as well as take geotagged photos. This tool is a great way to see which sensors your device supports and to also debug sensor related issues in your app. There is also a Windows Phone version of this app available as well. You can find this app in the Windows Store here: <http://bit.ly/1amWxRP>



Chapter Summary

In this chapter you were introduced to Sensor and Location platform in Windows Store apps. Here is a short summary of some key points from this chapter.

- Sensors are hardware components that are capable of measuring the physical outside world. Sensors can be accessed through the **Windows.Devices.Sensors** namespace.
- It is a good practice to remove any event handlers to sensors when the user control in which they are providing data to becomes no longer visible. This will free up the resources that are being used by the sensor. You can then re-add the event handlers when the user control is made visible again.
- Windows Store apps can access the location API through the **Windows.Devices.Geolocation** namespace.
- Using **async void** should be avoided when possible and should only be used in top level event handlers, in our case a button tapped event handler. Any code inside of a method that uses **async void** should be wrapped with a try/catch block. The main reason for this is that if an exception occurs in our code it won't be able to bubble up and out of the method. This will cause the application to crash.
- When using the location API, use the movement threshold and report interval values limit how often the position changed event is fired so as to conserve power by calculating locations only when needed.
- When using the location API enable the location capability in the app manifest.
- Geofencing is a new feature in Windows 8.1 that allows an app to define a geographical region which can be used to trigger events when the device running the app enters or exists the region.
- Once a **Geofence** object is created it can be added to the **Current.Geofences** property of the **GeofenceMonitor** class that is available inside apps. The **GeofenceMonitor** class is responsible maintaining information about the geofences and trigger the events on them as the device enters or leaves the geofences.
- The Visual Studio Simulator is one of the easiest ways to test geofences in your app.
- The Jack of Tools app is a must have app if you are working with sensors in your app. Get it from the Windows Store here: <http://bit.ly/1amWxRP>

Chapter 3: Bing Maps JavaScript API

The Bing Maps JavaScript SDK for Windows Store apps provides you with the great set of tools for adding mapping functionality to your JavaScript based Windows Store app. This API is nearly identical to the Bing Maps V7 AJAX SDK which is used for creating web based apps with Bing Maps. One big difference is that JavaScript files for this SDK are stored locally inside of your app. This helps your app start up faster as the JavaScript files can be bytecode cached by Windows 8.

Before you can make use of the Bing Maps SDK in a Windows Store application you have to first make sure you have it installed (<http://bit.ly/11TLdei>). You will also need a Bing Maps account and key as discussed in Chapter 1. Full documentation on the Bing Maps JavaScript API can be found here: <http://bit.ly/18zXRwQ>. You can also try out the Bing Maps Interactive SDK which is a great place to try out things in the JavaScript API quickly in a web browser: <http://bit.ly/1bKDFuD>.

Bing Maps Modular Framework

The Bing Maps V7 AJAX and Windows Store SDK's use a modular framework as a way to minimize loading of a bunch of features and functionalities that may not be needed. The base map control consists of a number of core features such as support for pushpins, polylines, polygons, and tile layers. Modules allow users to load only the features and functionalities they need, rather than loading everything up when the application starts. You can save yourself a lot of development time by using modules and avoid reinventing the wheel. Out of the box, Bing Maps provides the following modules:

Name	Description
Microsoft.Maps.AdvanceShapes	Adds support for complex polygons. i.e. polygons with holes.
Microsoft.Maps.Directions	Allows you to calculate a route and display it on the map. The route is draggable by default for easy customization. The instructions will also be nicely formatted.
Microsoft.Maps.Search	Provides an easy method for geocoding address and searching for points of interest from JavaScript.
Microsoft.Maps.Themes.BingTheme	Modifies the navigation bar, pushpin and infobox look and feel to match the Bing Maps consumer site (http://bing.com/maps).
Microsoft.Maps.Traffic	Adds a traffic incident and flow data to the map.
Microsoft.Maps.VenueMaps	Exposes the Venue Map functionality and can be used to find nearby venue maps and load them. Venue Maps are interactive buildings on the map that often show the layout of a building. For instance, you can load a venue map of a mall and see where all the stores are located.

In addition to the modules available through the Bing Maps API, there are a large number of community created modules are available through the Bing Maps V7 Modules CodePlex project (<http://bit.ly/1buEQ19>). This project has some great modules that provide useful functionalities such as client side heat maps, drawing tools, data import libraries and many more. However, not all community created modules are designed to work with Windows Store applications; you may need to make some modifications to get them to work with your code. You can also create custom modules, which is a really great way to promote code reuse.

Implementing Modules

Implementing modules is fairly easy and consists of the following three steps:

1. Add a reference to the **veapimodules.js** file from the Bing Maps SDK

2. Register the module if it isn't already registered
3. Load the module and run any post load logic

Adding a reference to the **veapimodules.js** file in the Bing Maps SDK simple requires loading in this script file after the standard **veapicore.js** file used by Bing Maps

```
<!-- Bing Map Control references -->
<script type="text/javascript"
    src="ms-appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>
<script type="text/javascript"
    src="ms-appx:///Bing.Maps.JavaScript/js/veapiModules.js"></script>
```

Before you can load a module, it must first be registered. All modules built into Bing Maps are already registered, however, if you are using a custom module you will need to register it. If you have used custom modules before in the web version of Bing Maps, be warned, registering modules in the Windows Store App version of the Bing Maps is done slightly differently. Registering custom modules is pretty straightforward; simply add a line of code to register the name and load in the JavaScript file containing your module code right after. You can do this by adding code, similar to the example below, into the head of the main html page.

```
<!-- Register code to our Custom Module -->
<script>Microsoft.Maps.registerModule('MyModule');

```

Now that the module is registered, we can load it. When to load a module really depends on the type of module you are using. For instance, the Bing Theme module needs to be loaded before the map is loaded, however the directions module doesn't need to be loaded until the user wants to get directions. Here is the basic way of loading a module:

```
Microsoft.Maps.loadModule('MyModule');
```

With some modules you may want to run code after the module has loaded. In this case, you can pass in a callback function as an option when loading the module. Doing the following will trigger a function called **myModuleLoaded** when the module has completed loading.

```
Microsoft.Maps.loadModule('MyModule', { callback: myModuleLoaded });
```

Throughout this chapter are several examples of how to use many of the built in modules in Bing Maps. In many of the upcoming chapters in this book we will see how to use many of custom modules available on the Bing Maps V7 Modules CodePlex project.

Creating custom modules

The Bing Maps Modular framework allows you to create reusable blocks of code that ties into Bing Maps. This saves on development time and is a great way to improve code quality by re-using proven and tested modules.

The basic overview of how to create a module

1. Create a single JavaScript file that contains all the code for your module.
2. Add a call to the Bing Maps **moduleLoaded** event to the end of the JavaScript file containing the code to for your module, passing in the name of your module. For example:

```
Microsoft.Maps.moduleLoaded('MyModule');
```

3. Register your module by adding a reference to where your modular plugin JavaScript file is located. This is often done right after the map is loaded. For example:

```
<!-- Register code to our Custom Module -->
```

```
<script>Microsoft.Maps.registerModule('MyModule');</script>
<script type="text/javascript" src="/js/myModule.js"></script>
```

4. Load your module by calling the **loadModule** method in Bing Maps and passing in a callback method that will be fired when the module has been loaded. For example:

```
Microsoft.Maps.loadModule('MyModule', { callback: myModuleLoaded });
```

Sample code of a basic module (i.e. mymodule.js)

```
var MyModule = function (map) {
    var localVariable = "";

    //Constructor
    function init() {

    }

    //Private Method
    function _doSomething() {
    }

    //Public method
    this.DoSomething = function () {
    };

    init();
};
Microsoft.Maps.moduleLoaded('MyModule');
```

Sample code of how to register and load a module (i.e. default.html)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Modules Example</title>

    <!-- WinJS references -->
    <link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
    <script src="//Microsoft.WinJS.2.0/js/base.js"></script>
    <script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

    <!-- BingMapsIntro references -->
    <link href="/css/default.css" rel="stylesheet" />
    <script src="/js/default.js"></script>

    <!-- Bing Map Control references -->
    <script type="text/javascript"
        src="ms-appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>
    <script type="text/javascript"
        src="ms-appx:///Bing.Maps.JavaScript/js/veapiModules.js"></script>

    <!-- Register code to our Custom Module -->
    <script>Microsoft.Maps.registerModule('MyModule');</script>
    <script type="text/javascript" src="/js/myModule.js"></script>

    <script type="text/javascript">
        (function () {
```



```

var map;

function initialize() {
    Microsoft.Maps.loadModule('Microsoft.Maps.Map', GetMap);
}

function GetMap() {
    map = new Microsoft.Maps.Map(document.getElementById("myMap"), {
        credentials: 'YOUR_BING_MAPS_KEY'
    });

    //Load the arrow module
    Microsoft.Maps.loadModule("MyModule", { callback: myModuleLoaded });
}

function myModuleLoaded() {
    //Implement post module load logic
}

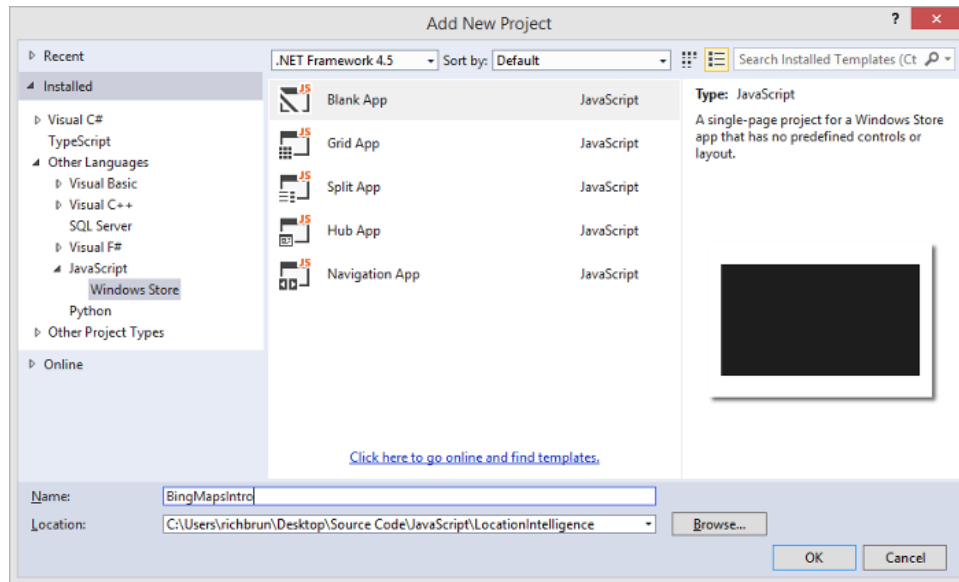
document.addEventListener("DOMContentLoaded", initialize, false);
})();
</script>
</head>
<body>
    <div id="myMap"></div>
</body>
</html>

```

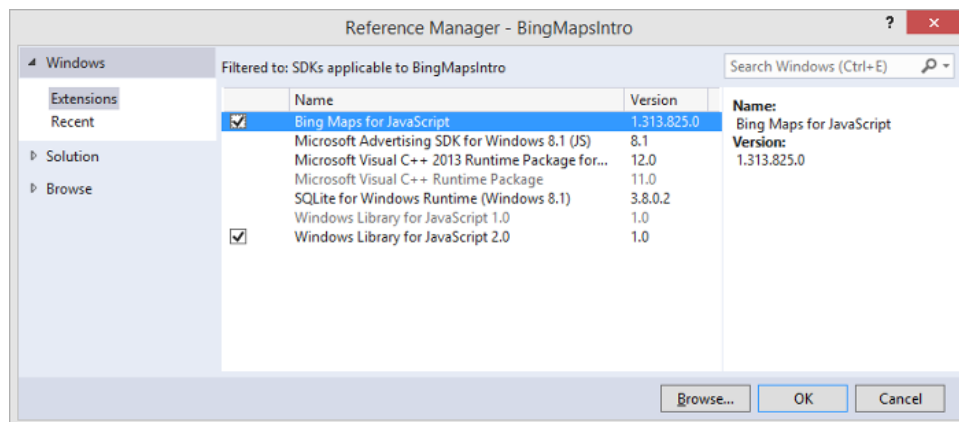
Note that for simplicity of the code sample the main JavaScript logic for loading the map and module were added to the head of an HTML document. As a best practice all JavaScript should be stored in a separate JavaScript file and added as a standard script reference in the head of the main HTML document. This will allow your application to make use of bytecode caching which allows your JavaScript files to load much faster.

Adding Bing Maps to a Windows Store app

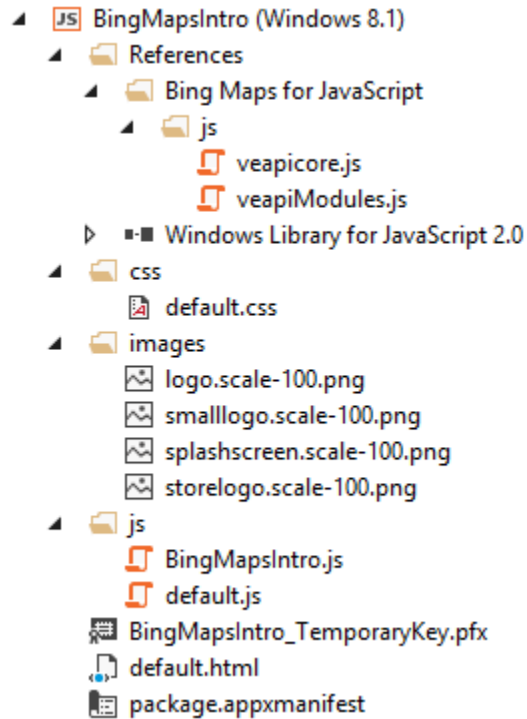
Open up Visual Studios and create a new project. In the window that opens, select **JavaScript -> Windows Store**. Select the Blank App template, and call the application **BingMapsIntro** and press OK.



Next add a reference to the Bing Maps SDK. To do this right click on the **References** folder and press **Add Reference**. Select **Windows -> Extensions**, and then select **Bing Maps for JavaScript**.



Next create a new JavaScript file by right clicking on the **js** folder and selecting **Add -> New item**. Call the file **BingMapsIntro.js**. We will use this file to put all the application logic into. At this point your application will look like this in the solution folder.



Notice that there are two JavaScript files inside of the Bing Maps for JavaScript reference. The **veapicore.js** file is the main JavaScript file that is needed to load a map in your application. The **veapiModules.js** file includes additional code that is needed when using Bing Maps modules with your map.

To add a map to the application, open the **default.html** file and add a script reference the Bing Maps core and modules JavaScript files and the **BingMapsIntro.js** file in the head section of the document. Next add a **div** to the body of the document where you want the map to be rendered. When you are done your HTML will look like this:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>BingMapsIntro</title>

  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
  <script src="//Microsoft.WinJS.2.0/js/base.js"></script>
  <script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

  <!-- BingMapsIntro references -->
  <link href="/css/default.css" rel="stylesheet" />
  <script src="/js/default.js"></script>

  <!-- Bing Map Control references -->
  <script type="text/javascript" src="ms-appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>
  <script type="text/javascript" src="ms-appx:///Bing.Maps.JavaScript/js/veapiModules.js"></script>

  <!-- Our Bing Maps JavaScript Code -->
  <script type="text/javascript" src="/js/BingMapsIntro.js"></script>
</head>
<body>
  <div id="myMap"></div>
```

```

</body>
</html>

```

Next open up the **BingMapsIntro.js** file and add the following code to load the map.

```

(function () {
    var map;

    function initialize() {
        Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: GetMap });
    }

    function GetMap() {
        map = new Microsoft.Maps.Map(document.getElementById("myMap"), {
            credentials: "YOUR_BING_MAPS_KEY"
        });
    }

    document.addEventListener("DOMContentLoaded", initialize, false);
})();

```

You can now build and run the application (F5) you will see your app open up with a full screen map of the world.



In the **default.html** file and add the following HTML after the map **div** control. This will add several buttons to the bottom app bar of the application which we will use to test out different features of the API. When the application is running you will simply need to swipe up from the bottom of the app to display these buttons.

```

<div id="appbar" data-win-control="WinJS.UI.AppBar"
    data-win-options="{layout:'custom', placement:'bottom'}">

    <button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'AddPushpinBtn', label: 'Add Pushpin', icon: '&#xE1C4;',
type:'button'}" />

```

```

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'AddCustomPushpinBtn', label: 'Add Custom Pushpin', icon:
'&#xE141;', type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'AddCustomHTMLPushpinBtn', label: 'Add HTML Pushpin',
icon: '&#xE2B1;', type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'AddPolylineBtn', label: 'Add Polyline', icon:
'&#xE199;', type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'AddPolygonBtn', label: 'Add Polygon', icon: '&#x2B1F;',
type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'AddAdvancedShapeBtn', label: 'Add Advance Shape', icon:
'&#xE191;', type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'AddInfoboxBtn', label: 'Add Infobox', icon: '&#xE134;',
type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'AddTileLayerBtn', label: 'Add Tile Layer', icon:
'&#xE154;', type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'ToggleTrafficBtn', label: 'Toggle Traffic', icon:
'&#xE0C3;', type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'GeocodeBtn', label: 'Geocode', icon: '&#xE11A;',
type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'ReverseGeocodeBtn', label: 'Reverse Geocode', icon:
'&#xE128;', type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'FindPoiBtn', label: 'Find POI', icon: '&#xE14D;',
type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'DirectionsBtn', label: 'Directions', icon: '&#xE1D1;',
type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'LoadVenueMapBtn', label: 'Load Venue Map', icon:
'&#xE10F;', type:'button'}" />

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'GpsBtn', label: 'GPS', icon: '&#xE2A1;', type:'button'}"
/>

<button data-win-control="WinJS.UI.AppBarCommand"
        data-win-options="{id: 'CustomModuleBtn', label: 'Custom Module', icon:
'&#xE115;', type:'button'}" />
</div>

```

In the **BingMapsIntro.js** file we will add the button event handlers. To do this we first will need to ensure the buttons are initialized by calling the **WinJS.UI.processAll** function, then we can get the buttons by Id and add a click event to them. Update the **initialize** function with the following code.

```
function initialize() {
    Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: GetMap });

    // Initialize WinJS controls.
    WinJS.UI.processAll();

    document.getElementById("AddPushpinBtn").addEventListener("click", AddPushpinBtn_Tapped,
true);

    document.getElementById("AddCustomPushpinBtn").addEventListener("click",
AddCustomPushpinBtn_Tapped, true);

    document.getElementById("AddCustomHTMLPushpinBtn").addEventListener("click",
AddCustomHTMLPushpinBtn_Tapped, true);

    document.getElementById("AddPolylineBtn").addEventListener("click",
AddPolylineBtn_Tapped, true);

    document.getElementById("AddPolygonBtn").addEventListener("click", AddPolygonBtn_Tapped,
true);

    document.getElementById("AddAdvancedShapeBtn").addEventListener("click",
AddAdvanceShapeBtn_Tapped, true);

    document.getElementById("AddInfoboxBtn").addEventListener("click", AddInfoboxBtn_Tapped,
true);

    document.getElementById("AddTileLayerBtn").addEventListener("click",
AddTileLayerBtn_Tapped, true);

    document.getElementById("ToggleTrafficBtn").addEventListener("click",
ToggleTrafficBtn_Tapped, true);

    document.getElementById("GeocodeBtn").addEventListener("click", GeocodeBtn_Tapped,
true);

    document.getElementById("ReverseGeocodeBtn").addEventListener("click",
ReverseGeocodeBtn_Tapped, true);

    document.getElementById("FindPoiBtn").addEventListener("click", FindPoiBtn_Tapped,
true);

    document.getElementById("DirectionsBtn").addEventListener("click", DirectionsBtn_Tapped,
true);

    document.getElementById("LoadVenueMapBtn").addEventListener("click",
LoadVenueMapBtn_Tapped, true);

    document.getElementById("GpsBtn").addEventListener("click", GpsBtn_Tapped, true);

    document.getElementById("CustomModuleBtn").addEventListener("click",
CustomModuleBtn_Tapped, true);
}
```

Next we need to create a bunch of functions that will be called by these button event handlers. Add the following functions after the **initialize** function.

```
function AddPushpinBtn_Tapped() {  
}  
  
function AddCustomPushpinBtn_Tapped() {  
}  
  
function AddCustomHTMLPushpinBtn_Tapped() {  
}  
  
function AddPolylineBtn_Tapped() {  
}  
  
function AddPolygonBtn_Tapped() {  
}  
  
function AddAdvanceShapeBtn_Tapped() {  
}  
  
function AddInfoboxBtn_Tapped() {  
}  
  
function AddTileLayerBtn_Tapped() {  
}  
  
function ToggleTrafficBtn_Tapped() {  
}  
  
function GeocodeBtn_Tapped() {  
}  
  
function ReverseGeocodeBtn_Tapped() {  
}  
  
function FindPoiBtn_Tapped() {  
}  
  
function DirectionsBtn_Tapped() {  
}  
  
function LoadVenueMapBtn_Tapped() {  
}  
  
function GpsBtn_Tapped() {  
}  
  
function CustomModuleBtn_Tapped() {  
}
```

If you run the application and swipe up from the bottom or right click you will see a black panel with all the buttons on it appear. Pressing any of the buttons won't do anything as we haven't added any logic to the event handlers yet. The following is a screenshot of the app bar when in portrait mode. As you can see the app bar buttons wrap around to fit into the available space.



Customizing the Map

You can customize how the map loads using a number of different map options. Some of these options can be specified when creating the map while others can be changed at any time. When loading the map you can specify settings from the **MapOptions** and **ViewOptions** class. When loading the map you must pass in a reference to DOM element where the map will be rendered. This is normally a reference to an empty **div**. If the **div** you want to load the map to has an id of **myMap** then you would load the map with code similar to this:

```
var map = new Microsoft.Maps.Map(document.getElementById("myMap"), {
    [MapOptions] or [ViewOptions]
});
```

Here are some of the more commonly used settings from the **MapOptions** class.

Name	Type	Description
backgroundColor	Microsoft.Maps.Color	The color to use for the map control background. This property can only be set when using the Map constructor.
credentials	string	The Bing Maps Key used to authenticate the application. This property can only be set when using the Map constructor.
disablePanning	boolean	A boolean value indicating whether to disable the user's ability to pan the map. The default value is false.
disableZooming	boolean	A boolean value indicating whether to disable the user's ability to zoom in or out. The default value is false.
enableClickableLogo	boolean	A boolean value indicating whether the Bing™ logo on the map is clickable. The default value is true. This property can only be set when using the Map constructor.
enableSearchLogo	boolean	A boolean value indicating whether to enable the Bing™ hovering search logo on the map. The default value is true. This property can only be set when using the Map constructor.
fixedMapPosition	boolean	A boolean indicating whether the div containing the map control is fixed on the page and the browser will not be resized. The default value is false. In this case the map control redraws if necessary based on any div or window resize. If this property is set to true, the map control does not check the size of the div containing it every time the map view changes, thus increasing the performance of the control. This property can only be set when using the Map constructor.

height	number	The height of the map. The default value is null. If no height is specified, the height of the div is used. If height is specified, then width must be specified as well.
inertialIntensity	number	A number between 0 and 1 specifying the intensity of the inertia animation effect. The inertia effect increases as the intensity value gets larger. The default value is .85. Setting this property to 0 indicates no inertia effect. The useInertia property must be set to true for the inertialIntensity value to have an effect.
showBreadcrumb	boolean	A boolean value indicating whether to display the "breadcrumb control". The breadcrumb control shows the current center location's geography hierarchy. For example, if the location center is Seattle, the breadcrumb control displays "World . United States . WA". The default value is false. The breadcrumb control displays best when the width of the map is at least 300 pixels. This property can only be set when using the Map constructor.
showDashboard	boolean	A boolean value indicating whether to show the map navigation control. The default value is true. This property can only be set when using the Map constructor.
showMapTypeSelector	boolean	A boolean value indicating whether to show the map type selector in the map navigation control. The default value is true. This property can only be set when using the Map constructor.
showScalebar	boolean	A boolean value indicating whether to show the scale bar. The default value is true. This property can only be set when using the Map constructor.
useInertia	boolean	A boolean value indicating whether to use the inertia animation effect during map navigation. The default value is true.
width	number	The width of the map. The default value is null. If no width is specified, the width of the div is used. If width is specified, then height must be specified as well.

Some of the map options can be changed after the map has loaded by using the **setOptions** function on the map. Here are some of the more commonly used settings from the **ViewOptions** class.




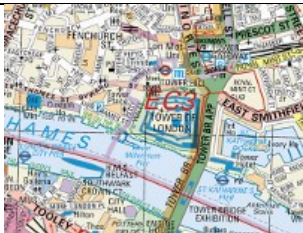

Name	Type	Description
animate	boolean	A boolean that specifies whether to animate map navigation. Note that this option is associated with each setView call and defaults to true if not specified.
bounds	Microsoft.Maps.LocationRect	The bounding rectangle of the map view. If both are specified, bounds takes precedence over center.
center	Microsoft.Maps.Location	The location of the center of the map view. If both are specified, bounds takes precedence over center.
centerOffset	Microsoft.Maps.Point	The amount the center is shifted. This property is ignored if center is not specified.
heading	number	The directional heading of the map. The heading is represented in geometric degrees with 0 or 360 = North, 90 = East, 180 = South, and 270 = West.
labelOverlay	Microsoft.Maps.LabelOverlay	A constant indicating how map labels are displayed.
mapTypeId	Microsoft.Maps.MapTypeId	The map type of the view.

padding	number	The amount of padding in pixels to be added to each side of the bounds of the map view.
zoom	number	The zoom level of the map view.

Any of these settings can be changed after the map has been loaded by using the **setView** function on the map.

MapTypeId Enumerator

This enumerator is used to specify the type of map style that should be displayed by the map and is specified as **Microsoft.Maps.MapTypeId.[Name]** where Name can be any of the following values.

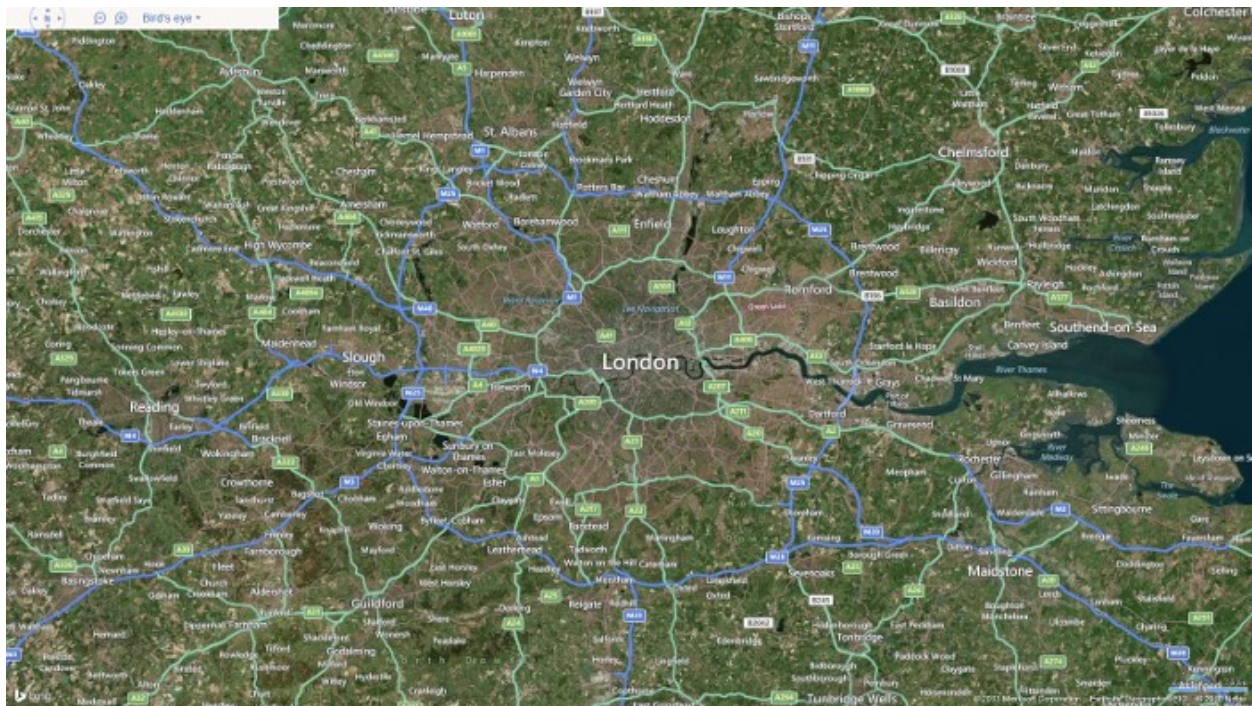
Name	Description	Example
aerial	The map displays aerial imagery.	
auto	The map is set to choose the best imagery for the current view.	
birdseye	The map displays an angled view of aerial imagery.	
collinsBart	The map displays Collins Bartholomew imagery. This imagery is only available in London, UK. When the map is panned or zoom out of range of this map imagery the road map imagery will be displayed. Map culture must be set to en-GB.	
mercator	The map does not display any imagery. Use this option if you want to display custom imagery instead of Bing Maps imagery.	
ordnanceSurvey	The map displays Ordnance Survey imagery. Ordnance Survey imagery is only available in the UK. Bing Maps provides the 1:25,000 OS Explorer Map and 1:50,000 OS Landranger maps. When the map is panned or zoom out of range of this map imagery the road map imagery will be displayed. Map culture must be set to en-GB.	
road	The map displays road imagery.	

Customizing the Map when Loading

As mentioned before you can customize the map when loading it. Use the following code to update how the map is loaded in the **GetMap** function such that it sets the type to aerial, the zoom level to 10, and the map center over London, UK (51.50632, -0.12714).

```
map = new Microsoft.Maps.Map(document.getElementById("myMap"), {
  credentials: "YOUR_BING_MAPS_KEY",
  center: new Microsoft.Maps.Location(51.50632, -0.12714),
  mapTypeId : Microsoft.Maps.MapTypeId.aerial,
  zoom: 10
});
```

This results in the map being loaded, zoomed in over London, UK with the aerial imagery being displayed.



The Breadcrumb Control

The breadcrumb control provides a hierarchy of locations based on where the map is centered over. This hierarchy usually takes the form of World -> Country -> Admin District -> City -> Venue Map Floor. Besides being a nice way to keep track of what you are looking at it also provides a useful way for jumping up a level to see a higher level view of an area. It also provides a method for switching between the floors of a venue map. To enable the breadcrumb simply set the **showBreadcrumb** map option to true. When enabled the bread crumb will be added as part of the navigation bar and look something like this:

... • ... • England • London

Culture and Localization

The Bing Maps JavaScript API supports 30 languages. By default the map control will automatically render in the language that the user's device is set to. This makes it easy for you as a developer as you don't need to worry about setting the language of the map. However, if you for some reason you would like to limit your application to one

language you can force the map to use a specific language by specifying the **culture** property when loading the **Microsoft.Maps.Map** module. A full list of supported cultures can be found here: <http://bit.ly/1d1Ke0Z>.

In addition to the **culture** property of the map there is also a **homeRegion** property. This property also defaults to the settings on the user's device but can be hard coded if desired. The home region property is used to determine how to handle geopolitical issues on the map. For instance, many countries recognize the Persian Gulf as a body of water in the Middle East, however many Arabic countries recognize this same body of water as the Arabian Gulf. As such the home region property allows the map to decide which name to use. You can hard code the home region by passing in a 2 letter country code as a string value. Not all regions are currently supported. If you come across a region that isn't supported the map may default to US. If you prefer to use a different home region you can simply hard code in a value that you feel comfortable using.

If you are curious about the Ordnance Survey and Collins Bartholomew maps styles available in Great Britain, you can set culture and home region as shown below to have them appear as options in the navigation bar.

```
Microsoft.Maps.loadModule('Microsoft.Maps.Map', {
  callback: GetMap,
  culture: 'en-GB',
  homeRegion: 'GB'
});
```

Map Function Methods

There are a lot of different function methods on the map which you can use. Here are some of the more common ones.

Name	Type	Description
getBounds	Microsoft.Maps.LocationRect	Returns the location rectangle that defines the boundaries of the current map view.
getCenter	Microsoft.Maps.Location	Returns the location of the center of the current map view.
getCredentials		Gets the session ID. This method takes in callback function and passes a session ID to it which can be used in place of a Bing Maps key when calling other Bing Maps services.
getHeading	number	Returns the heading of the current map view.
getHeight	number	Returns the height of the map control.
getMapTypeId	Microsoft.Maps.MapTypeId	Returns a string that represents the current map type displayed on the map. Valid map types are listed in the MapTypeId Enumeration topic.
getMetersPerPixel	number	Returns the current scale in meters per pixel of the center of the map.
getOptions	MapOptions	Returns the map options that have been set. Note that if an option is not set, then the default value for that option is assumed and getOptions returns undefined for that option.
getWidth	number	Returns the width of the map control.
getZoom	number	Returns the zoom level of the current map view.
getZoomRange	object:{min:number, max:number}	Returns the range of valid zoom levels for the current map view.
isRotationEnabled	boolean	Returns true if the current map type allows the heading to change; false if the display heading is fixed.
setMapType	Microsoft.Maps.MapTypeId	Sets the current map type. Takes in MapTypeId enumerator.

setOptions	object:{MapOptions}	Sets the height and width of the map.
setView	object:{ViewOptions}	Sets the map view based on the specified ViewOptions .
tryLocationToPixel	null, Microsoft.Maps.Point, or Microsoft.Maps.Point[]	<p>Converts a specified Location to a Point on the map relative to the specified PixelReference. If reference is not specified, PixelReference.viewport is used. If the map is not able to convert the Location, null is returned.</p> <p>Alternatively, it can also convert an array of Locations and return an array of Points. If any of the conversions fail, null is returned.</p>
tryPixelToLocation	null, Microsoft.Maps.Location, or Microsoft.Maps.Location[]	<p>Converts a specified Point to a Location on the map relative to the specified PixelReference. If reference is not specified, PixelReference.viewport is used. If the map is not able to convert the Point, null is returned.</p> <p>Alternatively, it can also convert an array of Points and return an array of Locations. If any of the conversions fail, null is returned.</p>

Changing the Map View

There are two different ways to change the map view. One way is to use the **setView** function on the map. The other is to use the **setMapType** function. Majority of the time you will likely only use the **setView** method as this also allows you to specify a map type in addition to other view settings. The most common method of changing the view that is used consists of setting the center and zoom level. The following line of code can be used to center the map over Barringer Crater located at coordinates 35.027222, -111.0225 and set the zoom level to 16.

```
map.setView({ center: new Microsoft.Maps.Location(35.027222, -111.0225), zoom: 16 });
```



One common scenario that developers come across is that they end up having an array of **Location** objects which they want to display on the map. They may mark these locations on the map using pushpins but that won't set the map view to show those locations. What you can do is use this array of locations and pass them into the **fromLocations** function on the **Microsoft.Maps.LocationRect** class. This function will return a **LocationRect** that encloses all the Location objects that were passed into it. This **LocationRect** can then be passed to the bounds setting when setting the map view. However, some developers may notice that this results in some pushpins being cut off at the maps edge. The reason for this is that the **fromLocations** function only calculates the bounding box based on the **Location** objects, and not on the additional area that the pushpin icons use. To accommodate this scenario the padding setting can be used to buffer the view by a number of pixels. Generally setting this value to twice the large width/height value of your pushpin icons works well. Here is a code sample that demonstrates how this can be done.

```
var locs = [array of Microsoft.Maps.Location];
var rect = Microsoft.Maps.LocationRect.fromLocations(locs);

map.setView({ bounds: rect, padding: 80 });
```

Overlaying Content on the Map

Maps have evolved from being a source of information to being a canvas that information is displayed on top of. This information can take many forms and can be represented in several different ways. The following sections show how to add a several different types of information to the map.

Locations, LocationRects and Colors

Like many API's we need a few basic classes to make everything work. The **Location** class will likely be one of the most used class in your mapping application. This class stores the coordinate information needed to mark locations on a map. The **Location** class consists of four properties; **latitude**, **longitude**, **altitude**, and **altitudeReference**. Latitude property is used to represent how far north or south a location is. This value is an angle measured around the center of the earth from the equator towards the poles. A positive value is in the northern hemisphere and a negative value is in the southern hemisphere. This value has a range of -90 to 90 degrees however due to the mathematics involved in representing the spherical globe as a flat 2D map results in some calculations approaching infinity. To avoid this, Bing Maps, and many other mapping platforms that use the Mercator projection system, clip the latitude coordinates to approximately -85 and 85 degrees. The **longitude** property stores the angle of horizontal offset from the prime meridian (0 degrees). This property has a value between -180 and 180 degrees. The **altitude** property is optional and stores the vertical distance the location is above a point in meters. The **altitudeReference** property is optional and stores information on what the altitude measurement is relative, for example; ground or ellipsoid. Note that the altitude information only has an effect on where things are displayed when the map is using the Birdseye map type. The following code shows two methods for creating **Location** objects.

```
var loc = new Microsoft.Maps.Location(latitude, longitude);

//With altitude information
var loc = new Microsoft.Maps.Location(latitude, longitude, altitude, altitudeReference);
```

The **LocationRect** class, also known as a bounding box, consists of a set of coordinates that are used to represent rectangular area on the map. These are often used for setting the map view but are also useful for doing calculations. A **LocationRect** object has three properties; **center**, **width** and **height**. The center property is a **Location** object marking the center of the **LocationRect** area. The **width** and **height** properties are angles in degrees from the center of the **LocationRect** to the edges. An instance of the **LocationRect** class can be created using the following code.

```
var rect = new Microsoft.Maps.LocationRect(center, width, height);
```

In addition to creating a **LocationRect** object using center, width and height values the following static function methods can be used to create a **LocationRect** as well:

Name	Usage	Description
fromCorners	Microsoft.Maps.LocationRect.fromCorners(north: Location, southeast: Location)	Returns a LocationRect using the specified locations for the northwest and southeast corners.
fromEdges	Microsoft.Maps.LocationRect.fromEdges(north:number, west:number, south:number, east:number, altitude:number, altitudeReference: AltitudeReference)	Returns a LocationRect using the specified northern and southern latitudes and western and eastern longitudes for the rectangle boundaries.
fromLocations	Microsoft.Maps.LocationRect.fromLocations(Microsoft.Maps.Location[])	Returns a LocationRect using a list of locations or an array of locations.
fromString	Microsoft.Maps.LocationRect.fromString("north,west,south,east")	Creates a LocationRect from a string with the following format: "north,west,south,east". North, west, south and east specify the coordinate number values.

The following is a list of common function methods that are part of the **LocationRect** class.

Name	Type	Description
contains	Boolean	Determines if a Location is within the LocationRect .
getEast	Number	Returns the longitude that defines the eastern edge of the LocationRect .
getNorth	Number	Returns the latitude that defines the northern edge of the LocationRect .
getNorthwest	Microsoft.Maps.Location	Returns the Location that defines the northwest corner of the LocationRect .
getSouth	Number	Returns the latitude that defines the southern edge of the LocationRect .
getSoutheast	Microsoft.Maps.Location	Returns the Location that defines the southeast corner of the LocationRect .
getWest	Number	Returns the latitude that defines the western edge of the LocationRect .
intersects	Boolean	Determines if one LocationRect intersects with this LocationRect .

Colors are used by Polylines and Polygons in Bing Maps and are represented by the **Microsoft.Maps.Color** class. This class has four properties; **a** (alpha), **r** (red), **g** (green), **b** (blue). These properties are defined as follows.

Name	Type	Description
a	number	The opacity of the color. The range of valid values is 0 to 255.
r	number	The red value of the color. The range of valid values is 0 to 255.
g	number	The green value of the color. The range of valid values is 0 to 255.
b	number	The blue value of the color. The range of valid values is 0 to 255.

Creating an instance of the Color class can be done in one of two ways. The first is to specify values for all the properties in the constructor. The second is to use the static function method **fromHex** which parses a string Hex color value into a Bing Maps Color. The following code shows how these two methods can be used to create a **Location** objects.

```
var color = new Microsoft.Maps.Color(150, 0, 255, 0);

//From a string Hex color
var color = Microsoft.Maps.Color.fromHex('#00FF00');
```

Layering Content

Content can be added to Bing Maps in a number of different ways. The main method of layering content on the map is by using **EntityCollection**'s. An **EntityCollection** is used to group together pushpins, polylines, polygons, tile layers, and Infoboxes as a collection of data to be displayed on the map. In addition to this it is also possible to other **EntityCollection**'s to an **EntityCollection** to create sub-collections. Here is a list of the most commonly used function methods of the **EntityCollection** class.

Name	Type	Description
clear		Removes all entities from the collection.
get	Microsoft.Maps.Pushpin, Microsoft.Maps.Polyline, Microsoft.Maps.Polygon, Microsoft.Maps.TileLayer, Microsoft.Maps.Infobox, Microsoft.Maps.EntityCollection	Takes in an index and returns the entity at the specified index in the collection.
getLength	number	Returns the number of entities in the collection.
getVisible	boolean	Returns whether the entity collection is visible on the map.
push	Microsoft.Maps.Pushpin, Microsoft.Maps.Polyline, Microsoft.Maps.Polygon, Microsoft.Maps.TileLayer, Microsoft.Maps.Infobox, Microsoft.Maps.EntityCollection	Adds the specified entity to the last position in the collection.
setOptions	object:{EntityCollectionOptions}	Sets the options for the entity collection.

The **entities** property on the map control derives from the **EntityCollection** class and is the main collection used to add content to the map. The following shows how to add an **EntityCollection** layer to the map.

```
var dataLayer = new Microsoft.Maps.EntityCollection();
map.entities.push(dataLayer);
```

Tip: It is a best practice to separate your data and infoboxes by using two layers. By doing this you will be able to ensure that your infobox is always rendered above the pushpins on the map regardless of the order in which they are added to the map.

Pushpins

Pushpins, sometimes also referred to as markers on other mapping platforms, are one of the primary ways of marking a location on a map. The **Pushpin** class has the following function methods.

Name	Type	Description
getAnchor	Microsoft.Maps.Point	Returns the point on the pushpin icon which is anchored to the pushpin location. An anchor of (0,0) is the top left corner of the icon.
getIcon	string	Returns the pushpin icon path.
getHeight	number	Returns the height of the pushpin, which is the height of the pushpin icon.
getLocation	Microsoft.Maps.Location	Returns the location of the pushpin.

getText	string	Returns the text associated with the pushpin.
getTextOffset	Microsoft.Maps.Point	Returns the amount the text is shifted from the pushpin icon.
getTypeName	string	The CSS class name used to style the pushpin.
getVisible	boolean	Returns a Boolean indicating whether the pushpin is visible or not. A value of false indicates that the pushpin is hidden, although it is still an entity on the map.
getWidth	number	Returns the width of the pushpin, which is the width of the pushpin icon.
setLocation	Microsoft.Maps.Location	Sets the location of the pushpin.
setOptions	object:{PushpinOptions}	Sets options for the pushpin.

When creating a pushpin a location must be passed as an argument to the constructor. Optionally an object consisting of pushpin options can also be passed as an argument. The following pushpin option properties can be used to create customized pushpins.

Name	Type	Description
anchor	Microsoft.Maps.Point	The point on the pushpin icon which is anchored to the pushpin location. An anchor of (0,0) is the top left corner of the icon. The default anchor is the bottom center of the icon.
draggable	boolean	A boolean indicating whether the pushpin can be dragged to a new position with the mouse or by touch.
height	number	The height of the pushpin, which is the height of the pushpin icon. The default value is 39.
htmlContent	string	HTML that is used to represent a pushpin.
icon	string	The path of the image to use as the pushpin icon.
text	string	The text associated with the pushpin.
textOffset	Microsoft.Maps.Point	The amount the text is shifted from the pushpin icon. The default value is (0,5).
typeName	string	A CSS class name used to style the pushpin.
visible	boolean	A boolean indicating whether to show or hide the pushpin. The default value is true. A value of false indicates that the pushpin is hidden, although it is still an entity on the map.
width	number	The width of the pushpin, which is the width of the pushpin icon. The default value is 25.

Adding pushpins to the map is pretty common task and easy to do. In the **BingMapsIntro.js** file update the **AddPushpinBtn_Tapped** button handler with the following code. This code will clear the map and add a new pushpin at the center of the map that has a text label set to 1.

```
function AddPushpinBtn_Tapped() {
    map.entities.clear();

    var center = map.getCenter();

    //Create Pushpin
    var pin = new Microsoft.Maps.Pushpin(center, {
        text : '1'
    });

    map.entities.push(pin);
}
```

If you click on this button a pushpin will appear at the center of the map.



The default pushpin is ok when testing but chances are you will want to change this to a different icon. If you simply want to change the color of the pushpin or use a completely different icon the process is the same. In either case you have to create a new image of the pushpin you want, add it to your project and then set the **icon** property of the pushpin options to the path of where the new pushpin image is. Take the following image and it to the images folder in the project (copy and paste from document or from sample code project).



sunny.png

Next update the **AddCustomPushpinBtn_Tapped** event handler with the following code. This code will use the sunny.png image as a pushpin. The width and height are passed in as options otherwise the image would be clipped to the dimensions of the default pushpin. An anchor is specified to center the image over the location it is representing otherwise it may appear as if the pushpin is moving away from the location when zooming.

```
function AddCustomPushpinBtn_Tapped() {
    map.entities.clear();

    var center = map.getCenter();

    //Create custom Pushpin
    var pin = new Microsoft.Maps.Pushpin(center, {
        icon: 'images/sunny.png',
        width: 45,
        height: 38,
        anchor: new Microsoft.Maps.Point(22, 16)
    });

    map.entities.push(pin);
}
```

If you click on this button a custom pushpin will appear at the center of the map.



Another option for creating custom pushpins is to pass in HTML into the **htmlContent** property of the pushpin options. Update the **AddCustomHTMLPushpinBtn_Tapped** event handler with the following code to create a custom HTML pushpin. This code will clear the map and add a new custom pushpin to the center of the map consisting of red colored text created in HTML.

```
function AddCustomHTMLPushpinBtn_Tapped() {
    map.entities.clear();

    var center = map.getCenter();

    //Create Pushpin
    var pin = new Microsoft.Maps.Pushpin(center, {
        htmlContent: '<span style="color:red;font-weight:bold;">Custom Pushpin</span>'
    });

    map.entities.push(pin);
}
```

If you click on this button a custom HTML pushpin will appear at the center of the map.



Tip: Another approach to creating custom pushpins is to use the HTML5 Canvas Pushpin module available here: <http://bit.ly/1i1hck4>. This module also you to create custom pushpins by drawing on the HTML5 canvas.

Polylines

Polylines, also known as LineStrings, allow you to draw connected lines on a map. The **Polyline** class has the following function methods.

Name	Type	Description
getLocations	Microsoft.Maps.Location[]	Returns the locations that define the polyline.
getStrokeColor	Microsoft.Maps.Color	Returns the color of the polyline.
getStrokeDashArray	String	Returns the string that represents the stroke/gap sequence used to draw the polyline.
getStrokeThickness	Number	Returns the thickness of the polyline.
getVisible	boolean	Returns whether the polyline is visible. A value of false indicates that the polyline is hidden, although it is still an entity on the map.
setLocations	Microsoft.Maps.Location[]	Sets the locations that define the polyline.
setOptions	object:{PolylineOptions}	Sets options for the polyline.

When creating a polyline an array of locations must be passed as an argument to the constructor. Optionally an object containing polyline options can also be passed as an argument. The following polyline option properties can be used to create customized polylines.

Name	Type	Description
strokeColor	Microsoft.Maps.Color	The color of the polyline.
strokeDashArray	string	A string representing the stroke/gap sequence to use to draw the polyline. The string must be in the format S, G, S*, G*, where S represents the stroke length and G represents gap length. Stroke lengths and gap lengths can be separated by commas or spaces. For example, a stroke dash string of "1 4 2 1" would draw the polyline with a dash, four spaces, two dashes, one space, and then repeated.
strokeThickness	number	The thickness of the polyline.

visible	boolean	A boolean indicating whether to show or hide the polyline. A value of false indicates that the polyline is hidden, although it is still an entity on the map.
---------	---------	---

To add a polyline to the map update the **AddPolylineBtn_Tapped** event handler with the following code. This code will clear the map and add a new polyline based on the center of the map that is red in color and has a width of 10 pixels.

```
function AddPolylineBtn_Tapped() {
    map.entities.clear();

    var center = map.getCenter();

    //Create array of locations
    var coords = [center, new Microsoft.Maps.Location(center.latitude + 1, center.longitude
+ 1)];

    //Create a polyline
    var line = new Microsoft.Maps.Polyline(coords, {
        strokeColor: new Microsoft.Maps.Color(150, 255, 0, 0),
        strokeThickness: 10
    });

    //Add the polyline to map
    map.entities.push(line);
}
```

If you click on this button a polyline that starts at the center of the map will be displayed.



Polygons

A polygon represents an area enclosed by a closed path (or loop), which is defined by a series of coordinates. The **Polygon** class is often used to draw out areas on a map. The **Polygon** class has the following function methods.

Name	Type	Description
getFillColor	Microsoft.Maps.Color	Returns the color of the inside of the polygon.

getLocations	Microsoft.Maps.Location[]	Returns the locations that define the polygon.
getStrokeColor	Microsoft.Maps.Color	Returns the color of the outline of the polygon.
getStrokeDashArray	string	Returns the string that represents the stroke/gap sequence used to draw the outline of the polygon.
getStrokeThickness	number	Returns the thickness of the outline of the polygon.
getVisible	boolean	Returns whether the polygon is visible. A value of false indicates that the polygon is hidden, although it is still an entity on the map.
setLocations	Microsoft.Maps.Location[]	Sets the locations that define the polygon.
setOptions	object:{PolygonOptions}	Sets options for the polygon.

When creating a polygon an array locations must be passed as an argument to the constructor. Optionally an object consisting of polygon options can also be passed as an argument. The following polygon option properties can be used to create customized polygons.

Name	Type	Description
fillColor	Microsoft.Maps.Color	The color of the inside of the polygon.
strokeColor	Microsoft.Maps.Color	The color of the outline of the polygon.
strokeDashArray	string	A string representing the stroke/gap sequence to use to draw the outline of the polygon. The string must be in the format S, G, S*, G*, where S represents the stroke length and G represents gap length. Stroke lengths and gap lengths can be separated by commas or spaces. For example, a stroke dash string of "1 4 2 1" would draw the polygon outline with a dash, four spaces, two dashes, one space, and then repeated.
strokeThickness	number	The thickness of the outline of the polygon.
visible	boolean	A boolean indicating whether to show or hide the polygon. A value of false indicates that the polygon is hidden, although it is still an entity on the map.

To add a polygon to the map update the **AddPolygonBtn_Tapped** event handler with the following code. This code will clear the map and add a new polygon based on the center of the map that is green in color.

```
function AddPolygonBtn_Tapped() {
    map.entities.clear();

    var center = map.getCenter();

    //Create array of locations
    var coords = [center];
    coords.push(new Microsoft.Maps.Location(center.latitude + 1, center.longitude + 1));
    coords.push(new Microsoft.Maps.Location(center.latitude - 1, center.longitude + 1));
    coords.push(center);

    //Create a polygon
    var polygon = new Microsoft.Maps.Polygon(coords, {
        fillColor: new Microsoft.Maps.Color(150, 0, 255, 0),
        strokeColor: new Microsoft.Maps.Color(150, 255, 0, 0),
        strokeThickness: 10
    });

    //Add the polygon to the map
    map.entities.push(polygon);
}
```

If you click on this button a polygon in the shape of a triangle will appear on the map.



Advance Shapes

The Polygon class in Bing Maps supports simple polygons that consist of a single exterior ring of coordinates. However, in more advance applications it is useful to be able to draw more complex polygons. Take for instance the borders of Lesotho, which is a land locked country within the main exterior borders of South Africa. In this case to properly represent this country's borders we would need to have a hole in the polygon.

Bing Maps has an Advance Shape module. This module contains a modified version of the **Polygon** class which takes in several arrays of **Locations** and supports polygons with holes. This module also contains a modified version of the **EntityCollection** which adds support for the modified **Polygon** class. This new Polygon class allows more complex polygons to be added to the map; such as those with holes in them. The advance **Polygon** class has many of the same function methods as the standard **Polygon** class and a couple of new ones. The following is a list of the new and modified function methods.

Name	Type	Description
getLocations	Microsoft.Maps.Location[]	Returns the location array that defines the first ring of the polygon.
getRings	[Microsoft.Maps.Location[]]	Returns an array of location arrays, where each location array defines a ring of the polygon.
setLocations	Microsoft.Maps.Location[]	Sets the locations that define a basic polygon.
setRings	[Microsoft.Maps.Location[]]	Sets an array of location arrays, where each location array defines a ring of the polygon.

To add a polygon to the map update the **AddAdvanceShapeBtn_Tapped** event handler with the following code. This code will clear the map and add a triangle polygon that has a triangular hole cut out of it.

```
function AddAdvanceShapeBtn_Tapped() {
    map.entities.clear();

    var rings = [
        [new Microsoft.Maps.Location(0, 0), new Microsoft.Maps.Location(20, 20), new
Microsoft.Maps.Location(0, 40), new Microsoft.Maps.Location(0, 0)],
```

```

    [new Microsoft.Maps.Location(5, 10), new Microsoft.Maps.Location(15, 20), new
Microsoft.Maps.Location(5, 30), new Microsoft.Maps.Location(5, 10)]
];

//Create a complex polygon
var polygon = new Microsoft.Maps.Polygon(rings, {
    fillColor: new Microsoft.Maps.Color(150, 0, 255, 0),
    strokeColor: new Microsoft.Maps.Color(150, 0, 0, 255)
});

//Add the polyon to the map.
map.entities.push(polygon);

//Set map view to display polygon
map.setView({bounds: Microsoft.Maps.LocationRect.fromLocations(rings[0])})
}

```

If you click on this button the complex polygon will appear on the map.



Infoboxes

An infobox, also sometimes refer to as an info window or popup, is a simple panel that displays information over top the map. This is often used to display information linked to a location after clicking on a pushpin. Here is a list of the most commonly used function methods in the **Infobox** class.

Name	Type	Description
getAnchor	Microsoft.Maps.Point	Returns the point on the infobox which is anchored to the map. An anchor of (0,0) is the top left corner of the infobox.
getDescription	string	Returns the string that is printed inside the infobox.
getHeight	number	Returns the height of the infobox.
getHtmlContent	string	Returns the info box as HTML.
getLocation	Microsoft.Maps.Location	Returns the location on the map where the infobox's anchor is attached.

getOffset	Microsoft.Maps.Point	Returns the amount the infobox pointer is shifted from the location of the infobox, or if showPointer is false, then it is the amount the infobox bottom left edge is shifted from the location of the infobox. The default value is (0,0), which means there is no offset.
getOptions	object:{InfoboxOptions}	Returns the infobox options.
getTitle	string	Returns a string that is the title of the info box.
getVisible	boolean	Returns whether the infobox is visible. A value of false indicates that the infobox is hidden, although it is still an entity on the map.
getWidth	number	Returns the width of the infobox.
setHtmlContent	string	Sets the HTML content of the infobox. You can use this method to change the look of the infobox. Note that infobox options are ignored if custom HTML is set. Also, when custom HTML is used to represent the info box, the infobox is anchored at the top-left corner.
setLocation	Microsoft.Maps.Location	Sets the location on the map where the anchor of the infobox is attached.
setOptions	object:{InfoboxOptions}	Sets options for the infobox.

When creating an infobox a location must be passed as an argument to the constructor. Optionally an object consisting of infobox options can also be passed as an argument. The following are the most common infobox option properties used to create customized infoboxes.

Name	Type	Description
description	string	The string displayed inside the infobox.
height	number	The height of the infobox. The default value is 126.
htmlContent	string	The HTML that represents the infobox. Note that infobox options are ignored if custom HTML is set. Also, if custom HTML is used to represent the infobox, the info box is anchored at the top-left corner.
location	Microsoft.Maps.Location	The location on the map where the infobox's anchor is attached.
offset	Microsoft.Maps.Point	The amount the info box pointer is shifted from the location of the infobox, or if showPointer is false, then it is the amount the info box bottom left edge is shifted from the location of the infobox. If custom HTML is set, it is the amount the top-left corner of the infobox is shifted from its location. The default offset value is (0,0), which means there is no offset.
showCloseButton	boolean	A boolean indicating whether to show the close dialog button on the infobox. The default value is true. By default the close button is displayed as an X in the top right corner of the infobox. This property is ignored if custom HTML is used to represent the infobox.
showPointer	boolean	A boolean indicating whether to display the infobox with a pointer. The default value is true. In this case the infobox is anchored at the bottom point of the pointer. If this property is set to false, the infobox is anchored at the bottom left corner. This property is ignored if custom HTML is used to represent the infobox.
title	string	The title of the infobox.

typeName	string	The CSS style name of the infobox. The default value is standard .
visible	boolean	A boolean indicating whether to show or hide the info box. The default value is true. A value of false indicates that the infobox is hidden, although it is still an entity on the map.
width	number	The width of the infobox. The default value is 256.

To add an infobox to the map update the **AddInfoboxBtn_Tapped** event handler with the following code. This code clears the map and adds a pushpin to it. This pushpin has a custom property added to it called Metadata that will contain title and description information for the location. A click event is added to the pushpin which is then used to add an infobox to display the metadata information.

```
function AddInfoboxBtn_Tapped() {
    map.entities.clear();

    var center = map.getCenter();

    //Create Pushpin
    var pin = new Microsoft.Maps.Pushpin(center);
    pin.Metadata = {
        title: 'My Pushpin Title',
        description: 'This is my description.'
    };

    //Add click event to pushpin to open infobox
    Microsoft.Maps.Events.addHandler(pin, 'click', OpenInfobox);

    //Add pushpin to the map.
    map.entities.push(pin);
}

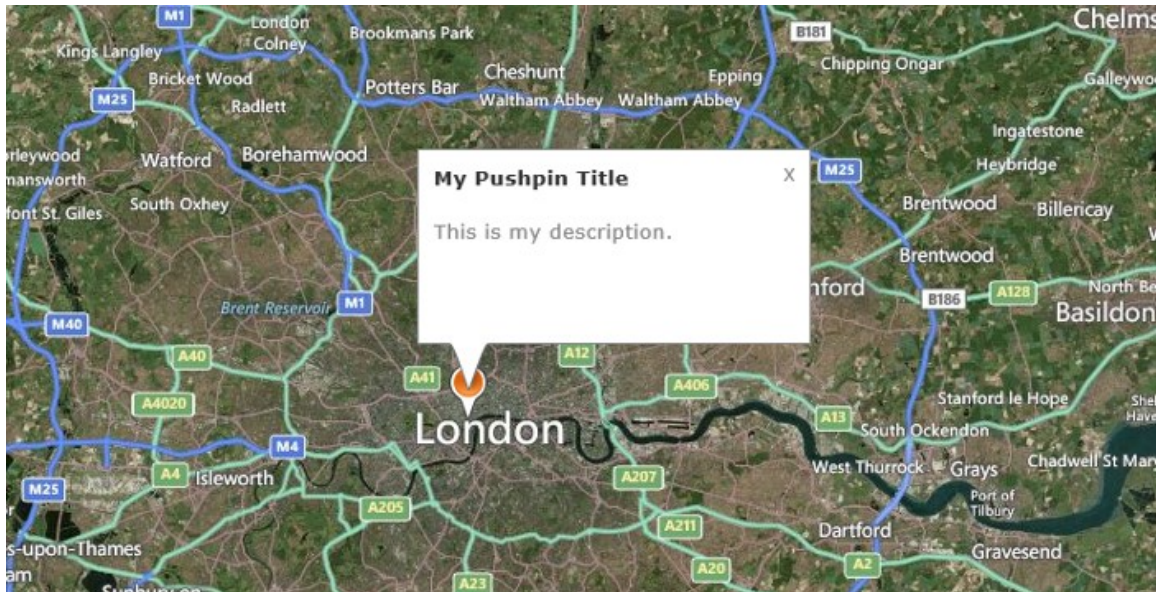
function OpenInfobox(e) {
    if (e.target && e.target.getLocation && e.target.Metadata) {
        //Get location of pushpin
        var loc = e.target.getLocation();

        //Create infobox options
        var infoboxOptions = {
            offset: new Microsoft.Maps.Point(0, 15),
            title: e.target.Metadata.title,
            description: e.target.Metadata.description,
        };

        //Create infobox
        var infobox = new Microsoft.Maps.Infobox(loc, infoboxOptions);

        //Add infobox to map
        map.entities.push(infobox);
    }
}
```

If you click on this button an infobox will appear at the center of the map.



Tip: When you have a lot of pushpins and only want to show one infobox at a time the best approach is to create one Infobox and to reuse it rather than creating an infobox for each pushpin. By doing this you will greatly reduce the number of DOM elements created by the application which will provide better performance. Also, separate your data from your infoboxes using layers to ensure your infoboxes always show up above your data. Take a look at this blog post for more information: <http://bit.ly/1daUGDd>.

Tile Layers

As we saw in Chapter 1, Bing Maps makes use of a tiling system to display maps of varying levels of details. It is possible to overlay custom tile layers on top of the map control. These layers can be used to replace the base maps completely, or overlaid on top the base maps to add an additional dimension to the data.

Tile layers are an excellent way to visualize large data sets on a map. There are two benefits to this approach. The first is that an image of the data is likely much smaller than the raw data itself, this means the map will be able to download the data faster. The second is that there will only ever be a few dozen tiles loaded on the map at the same time, the less objects the map control has to reposition when panning and zoom the better performance and the better the overall user experience will be.

The **TileLayer** class is used to define a tile layer in Bing Maps. The **TileLayer** class has the following function methods.

Name	Type	Description
getOpacity	number	Returns the opacity of the tile layer, defined as a double between 0 (not visible) and 1.
getTileSource	TileSource	Returns the tile source of the tile layer.
setOptions	None	Sets options for the tile layer.

When creating an instance of this class an object containing tile layer options can be passed to the constructor as an argument. The following are the most commonly used tile layer option properties.

Name	Type	Description
mercator	Microsoft.Maps.TileSource	The tile source for the tile layer.
opacity	number	The opacity of the tile layer, defined by a number between 0 (not visible) and 1.
visible	boolean	A boolean indicating whether to show or hide the tile layer. The default value is true. A value of false indicates

		that the tile layer is hidden, although it is still an entity on the map.
--	--	---

The **mercator** property takes in a **TileSource** object. The **TileSource** class contains all the information needed to define the source of the data for a tile layer. The **TileSource** class has the following function methods.

Name	Type	Description
getHeight	number	Returns the pixel height of each tile in the tile source.
getUriConstructor	string	Returns a string that constructs tile URLs used to retrieve tiles for the tile layer.
getWidth	number	Returns the pixel width of each tile in the tile source.

When creating an instance of **TileSource** class an object containing tile source options can be passed to the constructor as an argument. The following tile source option properties are available.

Name	Type	Description
height	number	The pixel height of each tile in the tile source. The default value is 256.
uriConstructor	string or function	A string or callback function that constructs the URLs used to retrieve tiles from the tile source. This property is required. When using a string the uriConstructor will replace {subdomain} and {quadkey}. When using a callback function an object will be passed as an argument to the has a x , y and levelOfDetail property.
width	number	The pixel width of each tile in the tile source. The default value is 256.

The simplest implementation of the **TileLayer** class is to set the **mercator** property to a **TileSource** that has a **uriConstructor** property set to a URL which uses a quadkey value to access map tiles as shown here.

```
//Create a tile layer source
var tileSource = new Microsoft.Maps.TileSource({
    uriConstructor: 'http://example.com/{quadkey}.png'
});

//Create a tile layer from the tile source
var tilelayer = new Microsoft.Maps.TileLayer({ mercator: tileSource });

//Add the tile layer to the map
map.entities.push(tilelayer);
```

Sometimes you may come across a tile service that uses the x, y, zoom level tile schema to access tiles. If this is the case you can pass a callback function to the **uriConstructor** property that generates the required tile URL. When this callback function is triggered the x, y, and zoom level values are provided for the given tile and can be used generate the required tile URL. For example, WaymarkedTrails.org uses the following URL format to provide access to hiking routes that are generated by OpenStreetMap data as a tile layer.

http://tile.waymarkedtrails.org/hiking/{z}/{x}/{y}.png

To add this tile layer to the map update the **AddTileLayerBtn_Tapped** event handler with the following code. This code will clear the map and add a new **TileLayer** that uses a callback function to get the URL to the hiking map using the x, y, and zoom level values.

```
function AddTileLayerBtn_Tapped()
{
    map.entities.clear();

    //Create a tile layer source
    var tileSource = new Microsoft.Maps.TileSource({ uriConstructor: getTilePath });
```

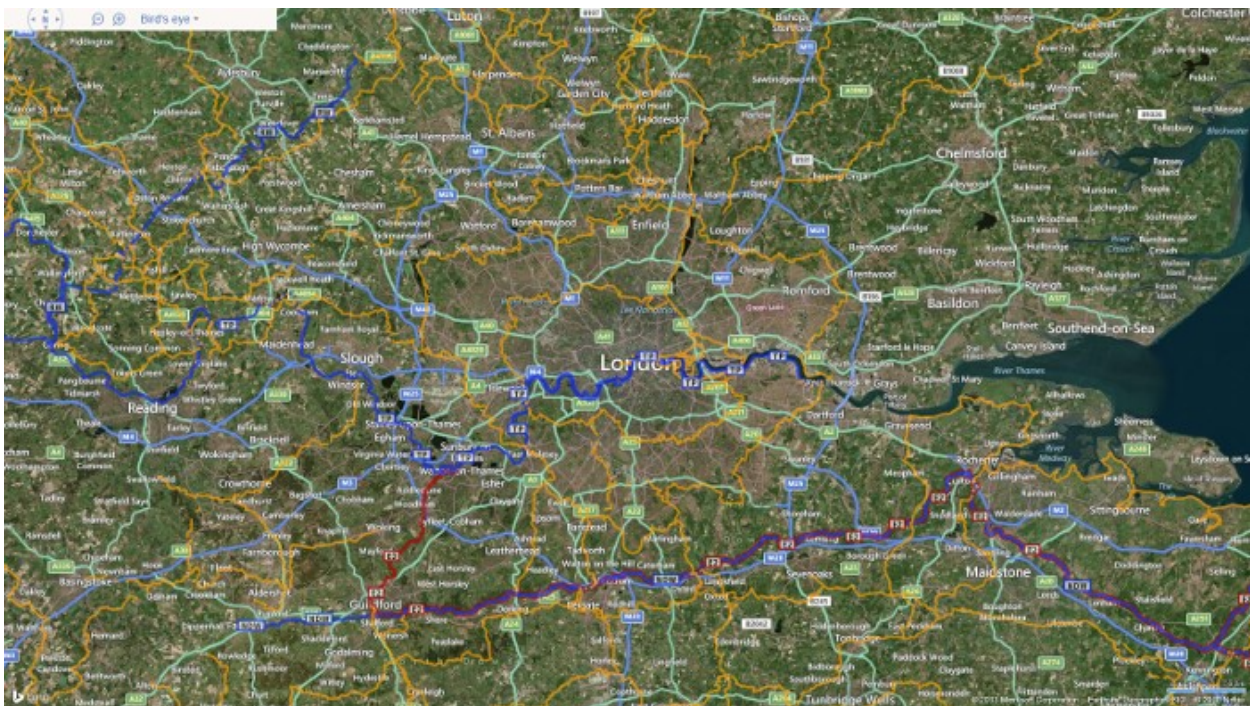


```
//Create a tile layer from the tile source
var tilelayer = new Microsoft.Maps.TileLayer({ mercator: tileSource, opacity: .7 });

//Add the tile layer to the map
map.entities.push(tilelayer);
}

function getTilePath(tile)
{
    //Tile Format http://tile.waymarkedtrails.org/hiking/{z}/{x}/{y}.png
    return 'http://tile.waymarkedtrails.org/hiking/' + tile.levelOfDetail + '/' + tile.x +
    '/' + tile.y + '.png';
}
```

If you click on this button you will see a bunch of colored lines that mark hiking trails appear on the map. Now if you pan and zoom the map you should find that the map moves smoothly, almost as if it didn't have any data at all on it.



There are many companies that expose mapping data in the form of a Web Mapping Service (WMS) or Web Map Tile Service (WMTS). These services can be used to generate tiles, however some of them may require additional information such as the bounding box of a tile. To ingest these types of services you can use the callback function method to generate the required tile URL. When the callback function is triggered you can calculate the bounding box of the tile and use this information to generate the required tile URL. The calculations are fairly complex. Here is an example of how to implement a WMS service that uses the bounding box of the tile in the URL:

```
function getTilePath(tile)
{
    var mapSize = Math.pow(2, tile.levelOfDetail);

    var west = ((tile.x * 360) / mapSize) - 180;
    var east = (((tile.x + 1) * 360) / mapSize) - 180;

    var efactor = Math.exp((0.5 - tile.y / mapSize) * 4 * Math.PI);
    var north = (Math.asin((efactor - 1) / (efactor + 1))) * (180 / Math.PI);
```

```

efactor = Math.exp((0.5 - (tile.y + 1) / mapSize) * 4 * Math.PI);
var south = (Math.asin((efactor - 1) / (efactor + 1))) * (180 / Math.PI);

return 'http://wms1.ccgis.de/cgi-bin/mapserv?map=/data/umn/germany/germany.map&&VERSION=1.1.1&REQUEST=GetMap&SERVICE=WMS&SRS=EPSG%3A4326&BBOX=' + west + ',' + south + ',' + east + ',' + north + '&WIDTH=256&HEIGHT=256&LAYERS=Topographie&format=image/png';
}

```

Here are some great tile layer sources that you may find useful:

- OpenStreetMap derived services - <http://bit.ly/1aCamOA>
- Data.gov - <http://1.usa.gov/15jGq5C>
- USGS - <http://bit.ly/15RbIDA>
- Geoserver - <http://bit.ly/14LnTve>

Traffic Manager

The Bing Maps SDK provides two types of traffic data through the Traffic module. The first type is a tile layer that shows traffic flow data. This highlights roads with different colors to indicate what the flow of traffic is like compared to the speed limits on those roads. The second type of traffic data is traffic incidents. Traffic incidents are point based data that represent things like road closures, accidents, and construction. The **TrafficManager** class has the following methods available:

Name	Type	Description
getIsOn	boolean	Returns a boolean value indicating if traffic data is being displayed or not.
hide		Hides all traffic data.
hideFlow		Hides the traffic flow layer.
hideIncidents		Hides all traffic incidents.
hideLegend		Hides the traffic legend.
show		Displays all traffic data.
showFlow		Displays the traffic flow layer.
showIncidents		Displays all traffic incidents.
showLegend		Displays the traffic legend.

When creating an instance of the **TrafficManager** class the map must be passed as an argument to the constructor. To add the functionality to toggle the traffic data on and off update the **ToggleTrafficBtn_Tapped** event handler with the following code. This code clears the map and then checks to see if the **trafficManager** variable is defined, if it is not defined it will load the Traffic module and create a new instance of the **TrafficManager** class and then re-toggle the traffic button. When the **trafficManager** variable is defined the **getIsOn** method will be used to check if the traffic data is displayed or not, and the **show** or **hide** method will be used accordingly.

```

var trafficManager;

function ToggleTrafficBtn_Tapped()
{
    map.entities.clear();

    if (trafficManager) {
        if (trafficManager.getIsOn()) {
            trafficManager.hide();
        } else {
            trafficManager.show();
        }
    }
}

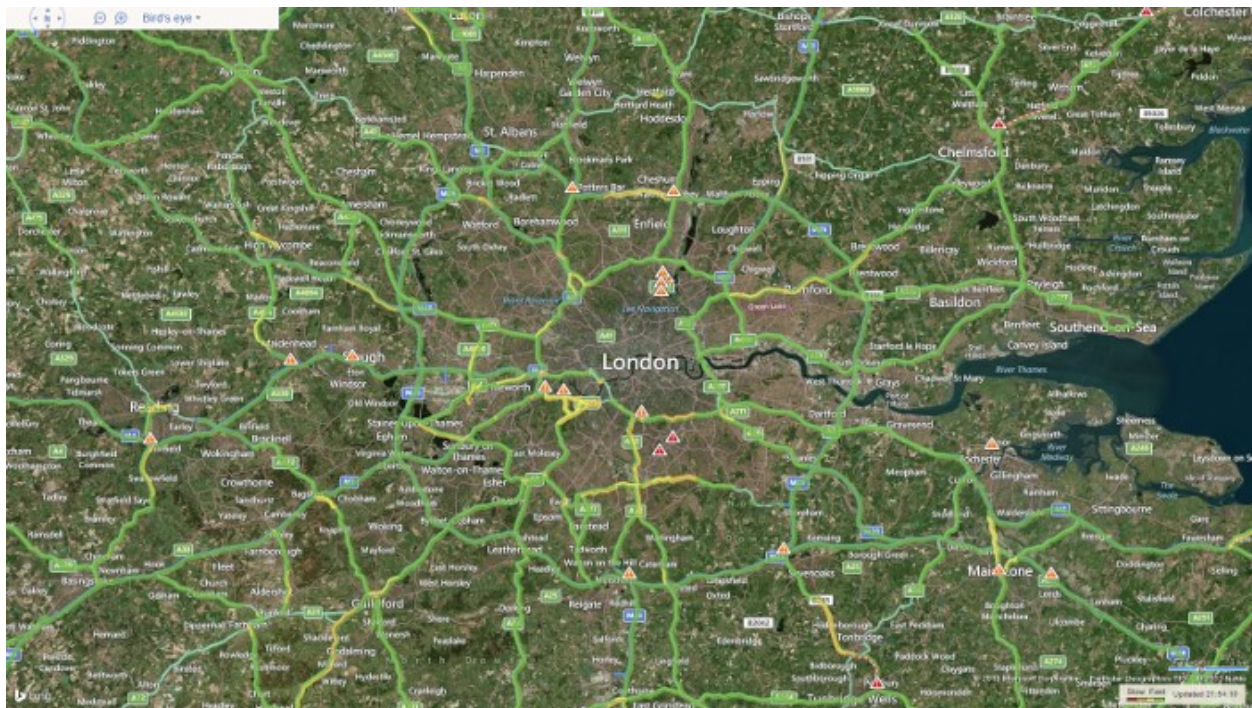
```

```

    }
  } else {
    //If traffic manager is not defined load the traffic module, create a
    //new instance of the traffic manager and call the toggle function again.
    Microsoft.Maps.loadModule('Microsoft.Maps.Traffic', {
      callback: function () {
        trafficManager = new Microsoft.Maps.Traffic.TrafficManager(map);
        ToggleTrafficBtn_Tapped();
      }
    });
  }
}

```

If you click on this button traffic data will appear on the map.



Traffic data is available in 34 countries. You can find a list of the currently supported regions here: <http://bit.ly/130ytgc>

Search Manager

The Bing Maps SDK provides the ability to geocode, reverse geocode and search for points of interest through the Search module. Geocoding is the process of taking an address and determining its location on the map. Reverse geocoding does the opposite and takes a coordinate and finds the closest known location. It does this by connecting to the Bing Maps REST services using an asynchronous method call. The **SearchManager** class is the primary class used to Search module. The **SearchManager** class has the following methods available:

Name	Description
geocode	Matches the address or place query in the specified request options to a location and returns the results to the request options callback function.
reverseGeocode	Matches the specified location to an address and returns the address results to the specified request options callback function.

search	Performs a search based on the specified request options and returns the results to the request options callback function.
--------	--

In order to geocode a location you need to pass in an object containing geocode request options into the **geocode** function method. The following are the most commonly used geocode request option properties.

Name	Type	Description
callback	function	A reference to a function to call when a successful result is returned from the geocode request. The callback function will receive a GeocodeResult object as an argument.
count	number	The maximum number of results to return. The maximum number that can be returned is 20.
errorCallback	function	A reference to a function to call when the request is returned with an error. The error callback function will receive an object containing the geocode request options used in the request.
where	string	A string containing the address or place to be matched to a location on the map. Required.

To add the functionality to geocode and address update the **GeocodeBtn_Tapped** event handler with the following code. This code clears the map and loads the search module if it isn't already loaded. It geocodes "New York, NY" then adds a pushpin to that location and sets the map view over the result.

```
var searchManager;

function GeocodeBtn_Tapped() {
    map.entities.clear();

    if (searchManager) {
        var searchRequest = {
            where: "New York, NY",
            callback: function (r) {
                if (r && r.results && r.results.length > 0) {
                    var pin = new Microsoft.Maps.Pushpin(r.results[0].location);
                    map.entities.push(pin);

                    map.setView({ center: r.results[0].location, zoom: 10 });
                }
            },
            errorCallback: function (e) {
                var dialog = new Windows.UI.Popups.MessageDialog("No results found.",
                    "Geocoding Result");
                dialog.showAsync();
            }
        };
        searchManager.geocode(searchRequest);
    } else {
        //If search manager is not defined load the search module, create a
        //new instance of the search manager and call the geocode function again.
        Microsoft.Maps.loadModule('Microsoft.Maps.Search', {
            callback: function () {
                searchManager = new Microsoft.Maps.Search.SearchManager(map);
                GeocodeBtn_Tapped();
            }
        });
    }
}
```


In order to reverse geocode a location you need to pass in an object containing reverse geocode request options into the **reverseGeocode** function method. The following are the most commonly used reverse geocode request option properties.

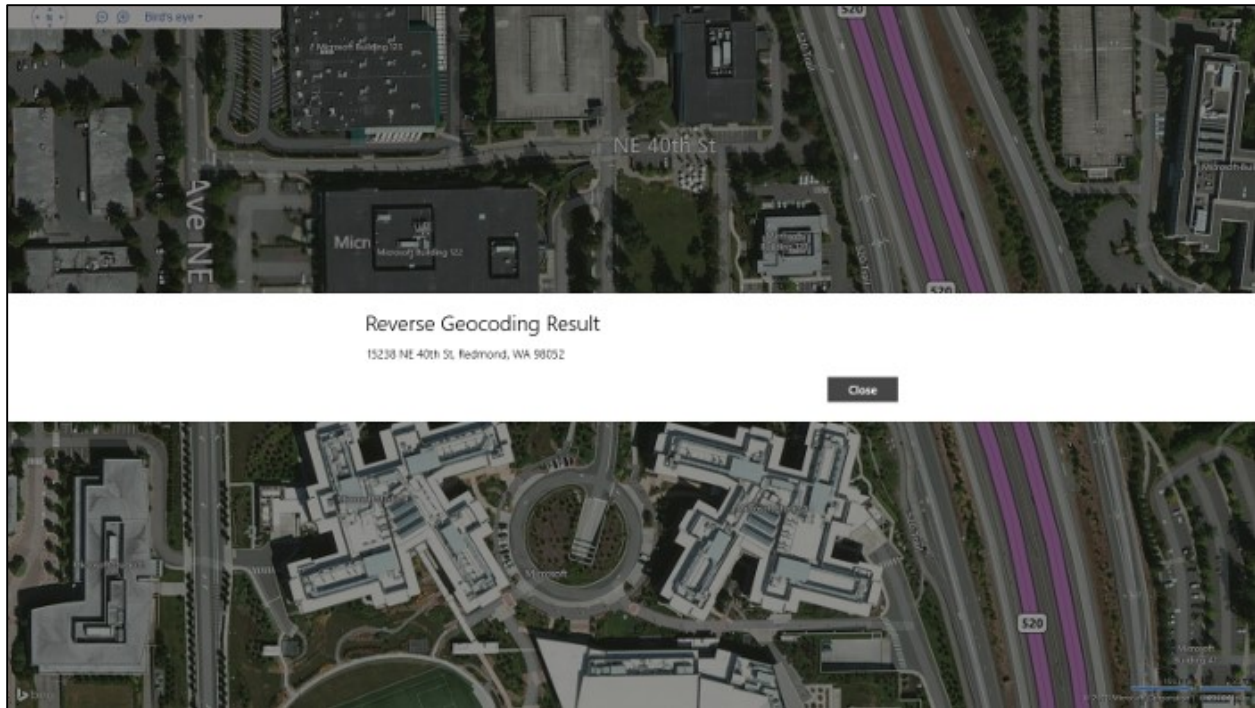
Name	Type	Description
callback	function	A reference to a function to call when a successful result is returned from the geocode request. The callback function will receive a PlaceResult object as an argument.
errorCallback	function	A reference to a function to call when the request is returned with an error. The error callback function will receive an object containing the geocode request options used in the request.
location	Microsoft.Maps.Location	The location to use to match to geographic entities and addresses.

To add the functionality to reverse geocode a coordinate update the **ReverseGeocodeBtn_Tapped** event handler with the following code. This code clears the map and loads the search module if it isn't already loaded. It then reverse geocodes the center of the map and if successful it will show a message on the screen with the display name (often an address) of the reverse geocoded location.

```
function ReverseGeocodeBtn_Tapped() {
    map.entities.clear();

    if (searchManager) {
        var searchRequest = {
            location: map.getCenter(),
            callback: function (r) {
                var dialog = new Windows.UI.Popups.MessageDialog(r.name, "Reverse Geocoding
Result");
                dialog.showAsync();
            },
            errorCallback: function (e) {
                var dialog = new Windows.UI.Popups.MessageDialog("Unable to reverse geocode
location.", "Reverse Geocoding Result");
                dialog.showAsync();
            }
        };
        searchManager.reverseGeocode(searchRequest);
    } else {
        //If search manager is not defined load the search module, create a
        //new instance of the search manager and call the reverse geocode function again.
        Microsoft.Maps.loadModule('Microsoft.Maps.Search', {
            callback: function () {
                searchManager = new Microsoft.Maps.Search.SearchManager(map);
                ReverseGeocodeBtn_Tapped();
            }
        });
    }
}
```

If you press the reverse geocode button over a location you will see a message be displayed with the display name of the location.



In order to find points of interest using the Search Manager you need to pass in an object containing search request options into the **search** function method. The following are the most commonly used search request option properties.

Name	Type	Description
callback	function	A reference to a function to call when a successful result is returned from the geocode request. The callback function will receive a SearchResponse object as an argument.
count	number	The maximum number of results to return. The maximum number that can be returned is 20.
errorCallback	function	A reference to a function to call when the request is returned with an error. The error callback function will receive an object containing the geocode request options used in the request.
query	string	The search string, such as "pizza in Seattle, WA". Either query or what/where needs to be specified. If both are specified, an error occurs.
startIndex	number	The index of the first result in the results. For example, if you had already returned a first set of 10 search results and now wanted to return the second set of 10 results, you would specify 10 as the startIndex and 10 as the count.
what	string	The business name, category, or other item for which the search is conducted. For example, "pizza" in the search string "pizza in Seattle".
where	string	The address, place name, or coordinates of the area for which the search is conducted. Pass coordinates as a string in the formation "latitude,longitude". For example, "Seattle" in the search string "pizza in Seattle".

To add the functionality to find nearby points of interest update the **FindPoiBtn_Tapped** event handler with the following code. This code clears the map and loads the search module if it isn't already loaded. It then uses the center of the map as the **where** parameter and sets the **what** parameter to "pizza". If successful all the results will be displayed on the map and the map view zoomed to all locations.

```
function FindPoiBtn_Tapped() {
```

```

map.entities.clear();

if (searchManager) {
    var center = map.getCenter();

    var searchRequest = {
        what: 'pizza',
        where: center.latitude.toFixed(5) + ',' + center.longitude.toFixed(5),
        count: 20,
        callback: function (r) {
            if (r && r.searchResults && r.searchResults.length > 0) {
                var locs = [];

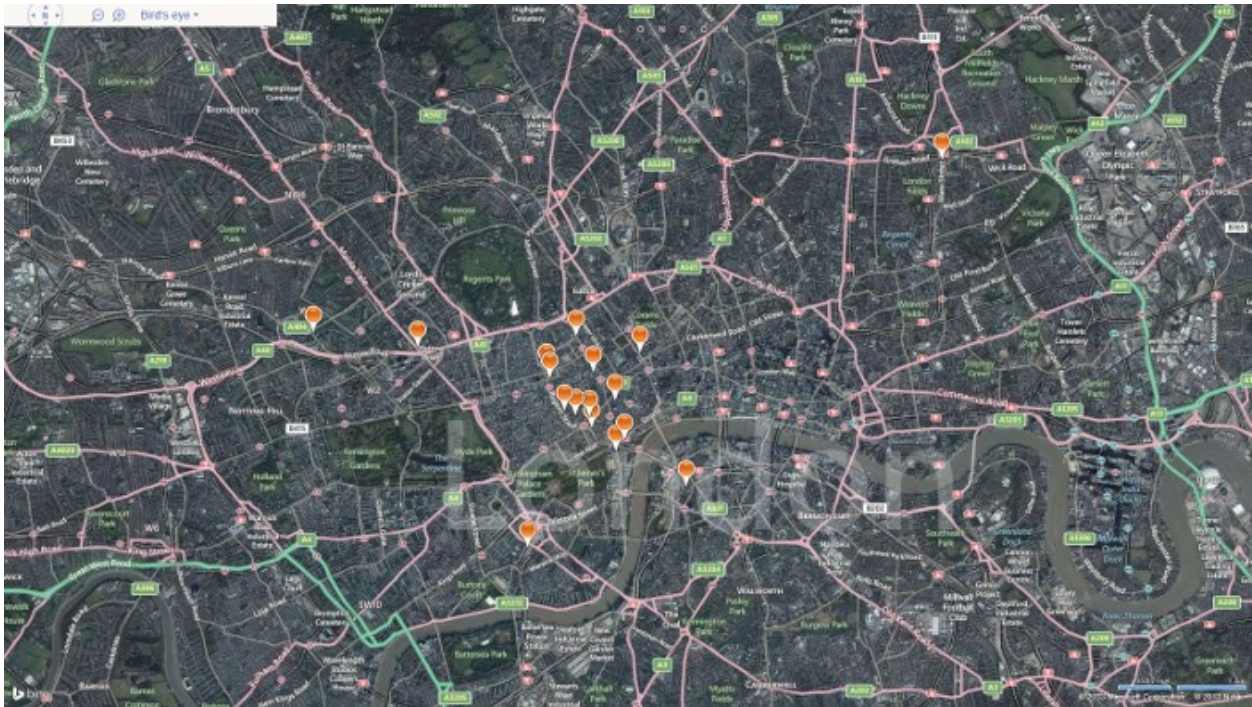
                for (var i = 0; i < r.searchResults.length; i++) {
                    var pin = new Microsoft.Maps.Pushpin(r.searchResults[i].location);
                    map.entities.push(pin);

                    locs.push(r.searchResults[i].location);
                }

                map.setView({ bounds: Microsoft.Maps.LocationRect.fromLocations(locs)
            });
        }
    },
    errorCallback: function (e) {
        var dialog = new Windows.UI.Popups.MessageDialog("Unable to find POIs.",
"POI Search Result");
        dialog.showAsync();
    }
};
searchManager.search(searchRequest);
} else {
    //If search manager is not defined load the search module, create a
    //new instance of the search manager and call the Find Poi function again.
    Microsoft.Maps.loadModule('Microsoft.Maps.Search', {
        callback: function () {
            searchManager = new Microsoft.Maps.Search.SearchManager(map);
            FindPoiBtn_Tapped();
        }
    });
}
}

```

If you press the find POI button it will search for points of interest near the center of the map that are related to the search term “pizza” and display them on the map.



Directions Manager

Directions can be calculated between multiple locations in the Bing Maps SDK by using the Directions module. This Directions module provides a JavaScript library that wraps the Bing Maps REST services to calculate directions by making asynchronous requests. The **DirectionsManager** class is the primary class used to calculate directions. Here is a list of the most commonly used function methods in the **DirectionsManager** class.

Name	Type	Description
addWaypoint	Microsoft.Maps.Directions.Waypoint	Adds a waypoint to the route at the given index, if specified. If an index is not specified the waypoint is added as the last waypoint in the route. The maximum number of walking or driving waypoints is 25. The maximum number of transit waypoints is 2. Up to 10 via points are allowed between two stop waypoints.
calculateDirections		Calculates directions based on request and render options set (setRequestOptions, setRenderOptions) and the waypoints added (addWaypoint). The directionsUpdated event fires when the calculation is complete and the route is displayed on the map. You must call this method after making any changes to the route options or waypoints.
clearDisplay		Clears the directions displayed and removes the route line from the map. This method does not remove waypoints from the route and retains all calculated direction information and option settings. To clear the calculated directions and options, use resetDirections.

getAllWaypoints	Microsoft.Maps.Directions.Waypoint[]	Returns the waypoints for the route.
getRenderOptions	Microsoft.Maps.Directions.DirectionsRenderOptions	Returns the route render options.
getRequestOptions	Microsoft.Maps.Directions.DirectionsRequestOptions	Returns the directions request options.
getRouteResult	Microsoft.Maps.Directions.Route[]	Returns the current calculated route(s). If the route was not successfully calculated, null is returned.
removeWaypoint		Removes the given waypoint or the waypoint identified by the given index from the route. Use calculateDirections to update the route once a waypoint has been removed.
resetDirections	object:{ResetDirectionsOptions}	If no options object is supplied, clears the directions request and render options and removes all waypoints.
setMapView		Sets the map view based on the current route index.
setRenderOptions	object:{DirectionsRenderOptions}	Sets the specified render options for the route.
setRequestOptions	object:{DirectionsRequestOptions}	Sets the specified route calculation options.

In order to calculate a route between locations you need to first add all the locations as **Waypoints** to the **DirectionsManager**. When creating an instance of the **Waypoint** class you must pass in an object containing **WayPointOptions**. Here is a list of the most commonly used waypoint options.

Name	Type	Description
address	string	The address string, business name, or search string of the waypoint. For example, the following strings are valid for this parameter: "Seattle", "Microsoft", "pizza", or "pizza Seattle". Either the address or location property must be specified.
isViaPoint	boolean	A boolean indicating whether the waypoint is a via point. A via point is a point along the route that is not a stop point. Set this property to true if you just want the route to pass through this location. The default value is false.
location	Microsoft.Maps.Location	The location of the waypoint. Either the address or location property must be specified.
pushpin	Microsoft.Maps.Pushpin	The custom pushpin used to represent this waypoint. This property overrides any pushpin options that apply to this waypoint that have been set using the direction render options.

You can set options for how to calculate the route by passing an object containing direction request options to the **setRequestOptions** function method of the **DirectionsManager**. Here is a list of the most commonly used direction request options.

Name	Type	Description
avoidTraffic	boolean	A boolean indicating whether to return traffic info when calculating the route. The default value is false.
distanceUnit	Microsoft.Maps.Directions.DistanceUnit	The unit to use when displaying direction distances. The default value is based on the specified culture. The options are

		Microsoft.Maps.Directions.DistanceUnit.kilometers or Microsoft.Maps.Directions.DistanceUnit.miles.
maxRoutes	number	The number of routes to calculate. If the routeMode is driving or walking, only 1 route is supported. If the routeMode is transit, up to 6 routes can be calculated and the default is 3.
routeAvoidance	Microsoft.Maps.Directions.RouteAvoidance[]	The features to avoid when calculating the route. The default value is none.
routeDraggable	boolean	A boolean indicating whether the route line on the map can be dragged with the mouse cursor. The default value is true.
routeIndex	number	If multiple routes are returned, the index of the route and directions to display. This property only applies to routes where the routeMode is transit, and in this case up to 6 routes are supported.
routeMode	Microsoft.Maps.Directions.RouteMode	The type of directions to calculate. The default value is driving.
routeOptimization	Microsoft.Maps.Directions.RouteOptimization	The optimization setting for the route calculation. The default value is shortestTime.
transitOptions	object:{TransitOptions}	The extra options for calculating a route if the routeMode is transit.

The **Microsoft.Maps.Directions.RouteOptimization** enumerator has the following options.

Name	Description
driving	Driving directions are calculated.
transit	Transit directions are calculated.
walking	Walking directions are calculated.

The **Microsoft.Maps.Directions.RouteAvoidance** enumerator has the following options.

Name	Description
none	Calculate the best route using any travel option available.
minimizeLimitedAccessHighway	Reduce the use of limited access highways when calculating the route.
minimizeToll	Reduce the use of roads with tolls when calculating the route.
avoidLimitedAccessHighway	Avoid using limited access highways when calculating the route.
avoidToll	Avoid using roads with tolls when calculating the route.
avoidExpressTrain	Avoid using express trains when calculating the route. This option only affects routes with a transitRouteMode that have the culture set to ja-jp.
avoidAirline	Avoid using airlines when calculating the route. This option only affects routes with a transitRouteMode that have the culture set to ja-jp.
avoidBulletTrain	Avoid using bullet trains when calculating the route. This option only affects routes with a transitRouteMode that have the culture set to ja-jp.

The **Microsoft.Maps.Directions.RouteOptimization** enumerator has the following options.

Name	Description
shortestTime	Calculate a route the shortest time.
shortestDistance	Calculate a route with the shortest distance.
minimizeMoney	Minimize the cost when calculating directions. This option only affects routes with a transitRouteMode that have the culture set to ja-jp.
minimizeTransfers	Minimize transit transfers when calculating directions. This option only affects routes with a transitRouteMode that have the culture set to ja-jp.

minimizeWalking	Minimize the amount of walking when calculating directions. This option only affects routes with a transitRouteMode that have the culture set to ja-jp.
-----------------	--

You can set options for how the route is rendered by passing an object containing direction render options to the **setRenderOptions** function method of the **DirectionsManager**. Here is a list of the most commonly used direction render options.

Name	Type	Description
autoDisplayDisambiguation	boolean	A boolean indicating whether to automatically display a disambiguation dialog for waypoints. The default value is true. If this value is set to true, a directionsError event caused by waypoint disambiguation is not thrown.
autoUpdateMapView	boolean	A boolean indicating whether to automatically set the map view to the best map view of the calculated route. The default value is true.
displayManeuverIcons	boolean	A boolean indicating whether to display the icons for each direction maneuver. The default value is true.
displayPostItineraryItemHints	boolean	A boolean indicating whether to display direction hints that describe when a direction step was missed. The default value is true.
displayPreliminaryItemHints	boolean	A boolean indicating whether to display direction hints that describe what to look for before you come to the next direction step. The default value is true.
displayRouteSelector	boolean	A boolean indicating whether to display the route selector control. The default value is true.
displayStepWarnings	boolean	A boolean indicating whether to display direction warnings. The default value is true.
displayTrafficAvoidanceOption	boolean	A boolean indicating whether to display the control that allows the user to choose to avoid traffic. The default value is true.
displayWalkingWarning	boolean	A boolean indicating whether to display a warning about walking directions. The default value is true.
drivingPolylineOptions	object:{PolylineOptions}	The options that define how to draw the route line on the map, if the RouteMode is driving.
itineraryContainer	HTMLElement	The DOM element inside which the directions itinerary will be rendered.
lastWaypointIcon	string	The URL of the icon to use for the end waypoint.
middleWaypointIcon	string	The URL of the icon to use for intermediate waypoints.
stepPushpinOptions	object:{PushpinOptions}	The options that define the pushpin to use for each direction step.
transitPolylineOptions	object:{PolylineOptions}	The options that define how to draw the route line on the map, if the RouteMode is transit.
viapointPushpinsOptions	object:{PushpinOptions}	The options that define the pushpin to use for each via point of the route, which are any waypoints other than the start and end waypoints.
walkingPolylineOptions	object:{PolylineOptions}	The options that define how to draw the route line on the map, if the RouteMode is walking.
waypointPushpinOptions	object:{PushpinOptions}	The options that define the pushpin to use for the route waypoint.

To add the functionality to calculate a simple route update the **DirectionsBtn_Tapped** event handler with the following code. This code clears the map and loads the directions module if it isn't already loaded. It then uses calculates the shortest by time driving directions from "New York, NY" to "Boston, MA".

```
var directionsManager;

function DirectionsBtn_Tapped() {
    map.entities.clear();

    if (directionsManager) {
        //Clear any previous directions information
        directionsManager.resetDirections();

        //Specify waypoints to navigate route through
        var startWaypoint = new Microsoft.Maps.Directions.Waypoint({ address: "New York, NY"
    });
        var endWaypoint = new Microsoft.Maps.Directions.Waypoint({ address: "Boston, MA" });

        directionsManager.addWaypoint(startWaypoint);
        directionsManager.addWaypoint(endWaypoint);

        //Set directions request options.
        var directionsOptions = {
            routeMode: Microsoft.Maps.Directions.RouteMode.driving,
            routeOptimization: Microsoft.Maps.Directions.RouteOptimization.shortestTime,
            distanceUnit: Microsoft.Maps.Directions.DistanceUnit.miles
        };

        directionsManager.setRequestOptions(directionsOptions);

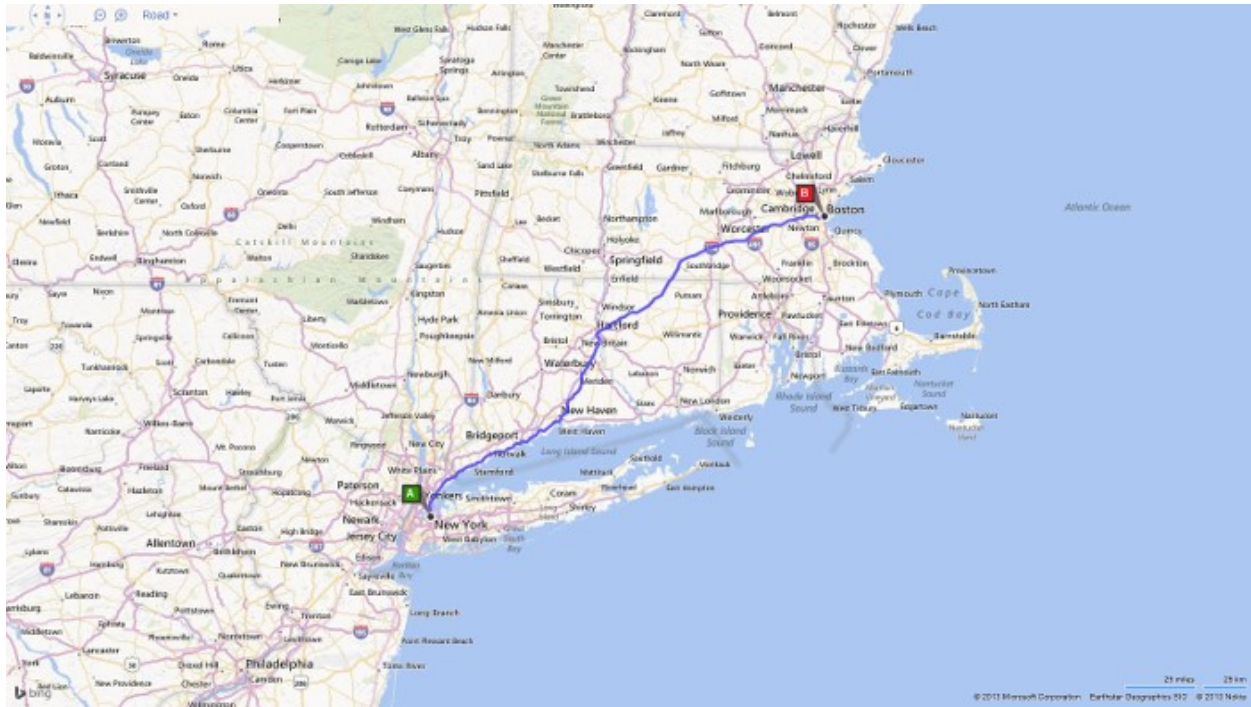
        //Calculate directions
        directionsManager.calculateDirections();

    } else {
        //If directions manager is not defined load the directions module, create a
        //new instance of the directions manager and call the directions function again.
        Microsoft.Maps.loadModule('Microsoft.Maps.Directions', {
            callback: function () {
                directionsManager = new Microsoft.Maps.Directions.DirectionsManager(map);

                // Specify a handler for when an error occurs when calculating directions
                Microsoft.Maps.Events.addHandler(directionsManager, 'directionsError',
function (e) {
                    var dialog = new Windows.UI.Popups.MessageDialog("Unable to calculate
route.");
                    dialog.showAsync();
                });

                DirectionsBtn_Tapped();
            }
        });
    }
}
```

If you press the directions button it will display a route from New York to Boston on the map.



Venue Maps

Within Bing Maps you have the ability to view Venue Maps. Venue Maps are often thought of as maps of indoor structures such as malls and airports; however venue maps can be of just about any complex structure that is spread out over a large area. Some other types of venue maps available in Bing Maps include: the layout of shopping districts, stadiums, race courses, parks, and universities. Venue Maps are only displayed when the **MapType** is set to road imagery. Venues are accessed through the Venue Maps module. The **VenueMapFactory** class is the primary class used for finding and rendering venues. Here is a list of the function methods available in this class.

Name	Type	Description
create	option:{VenueMapCreationOptions}	Creates a venue map. If the venue map is created successfully, a VenueMap is returned to the function specified in the success property of the venue map creation options. You can display a venue map using the show method of the VenueMap Class.
getNearbyVenues	option:{NearbyVenueMapOptions}	Searches for venue maps within a specified distance in meters. The callback function must accept a NearbyVenue array.

When finding nearby venues you must pass in an object containing nearby venue map options to the **getNearbyVenues** function method of the **VenueMapsFactory** class. Here is a list of the most commonly used nearby venue map options.

Name	Type	Description
callback	function	The function to call when the search is completed. The function must accept an array of VenueMap objects.
location	Microsoft.Maps.Location	The center of the circle in which to search for nearby venue maps.
radius	number	The radius, in meters, of the circle in which to search for nearby venue maps.

In order to display venue maps you must pass in an object containing venue map creation options to the **create** function method of the **VenueMapsFactory** class. Here is a list of the most commonly used venue map creation options.

Name	Type	Description
error	function	The function to call if the venue map was not successfully created. The function must accept two parameters: an integer which is an error code and an object that contains the arguments passed to the create method of the VenueMapFactory . The error codes are: <ol style="list-style-type: none"> 1 The metadata needed to create the venue map was not found because of a 404 or other web exception, or because the metadata was found but was empty. 2 The venue map metadata is invalid. 3 A timeout has occurred trying to retrieve the venue map metadata.
success	function	The function to call if the venue map was successfully created. The function must accept two parameters: a VenueMap and an object that contains the arguments passed to the create method of the VenueMapFactory .
venueMapId	string	A string that identifies the venue map to display.

There are a number of different ways to get a venue to display. One of the most common methods is to find nearby venues using the **getNearbyVenues** function method of the **VenueMapsFactory** class. This method returns an array of **NearbyVenue** objects. Nearby venues have a property called **metadata** that contains all the metadata for the venue such as the name, location, and unique venue id. The venue id is stored in the **MapID** property of the **metadata** object and can be used passed to **create** function method of the **VenueMapFactory** class to retrieve the corresponding **VenueMap** object. This venue object can then be displayed calling the show function method on the **VenueMap** once it is created. Update the **LoadVenueMapBtn_Tapped** event handler with the following code. It will search within a 1,000 meters of the center of the map for a venue and load the first one that is found.

```
var venueMapFactory;

function LoadVenueMapBtn_Tapped() {
    map.entities.clear();

    if (venueMapFactory) {
        var nearbyVenueOptions = {
            location: map.getCenter(),
            radius: 1000,
            callback: function (venues) {
                if(venues && venues.length > 0)
                {
                    var m = venues[0].metadata;

                    //Create venue map
                    venueMapFactory.create({
                        venueMapId: m.MapId,
                        success: function (venue) {
                            venue.show();

                            //Set the map type to road as Venue Maps are only displayed in road view.
                            map.setView({
                                mapTypeId: Microsoft.Maps.MapTypeId.road,
```

```

        center: venue.center,
        zoom: 17
    });
    }
    });
}
}
};

//Search for venues that are within 1000 meters (1km) of the center of the map.
venueMapFactory.getNearbyVenues(nearbyVenueOptions);

} else {
    //If venue map factory is not defined load the venue map module, create a
    //new instance of the venue map factory and call the venue map function again.
    Microsoft.Maps.loadModule('Microsoft.Maps.VenueMaps', {
        callback: function () {
            venueMapFactory = new Microsoft.Maps.VenueMaps.VenueMapFactory(map);
            LoadVenueMapBtn_Tapped();
        }
    });
}
}
}

```

If the venue map button is pressed and a venue exists near the center of the map, the map will zoom into the venue and the inside of the venue map will be displayed.



Working with Map Events

The Bing Maps control provides many events to allow your application to respond to user actions. The **EntityCollection**, **Map**, **Pushpin**, **Polyline**, and **Polygon** classes along with many of the modules all have events. Here are a couple of examples of where you may use events:

- If a user clicks on a shape, trigger an event that opens an infobox.
- If the map style changes you may want to change the color of items overlaid on the map so that they stand out better. Light colors on the aerial maps and dark colors on the road maps.
- If the user has moved the map you may want to load in new data for the current map view once the user has finished moving the map.

Events can be added and removed by using the function methods available in the **Microsoft.Maps.Events** class. The **Events** class has the following static methods available.

Name	Type	Description
addHandler	object	Attaches the handler for the event that is thrown by the target. Use the return object to remove the handler using the removeHandler method.
addThrottledHandler	object	Attaches the handler for the event that is thrown by the target, where the minimum interval between events (in milliseconds) is specified as a parameter.
hasHandler	boolean	Checks if the target has any attached event handler.
invoke		Invokes an event on the target. This causes all handlers for the specified event name to be called.
removeHandler		Detaches the specified handler from the event. The handlerId is returned by the addHandler and addThrottledHandler methods.

The following code sample demonstrates how to add and remove events. The target is the object you want to add the event to (i.e. the map), the **eventName** is the name of the event to attach as a string, and the handler is a callback function that is called when the event is triggered. When an event is added the **addHandler** function method returns an object (**handlerId**). This object can be passed to the **removeHandler** method later to detach the event from the object.

```
//Attach an event
var handlerId = Microsoft.Maps.Events.addHandler(target, 'eventName', handler);

//Remove an event
Microsoft.Maps.Events.removeHandler(handlerId);
```

There are a number of events available to the map, here are the most commonly used events:

Name	Description
click	Occurs when the mouse is used to click the map.
dblclick	Occurs when the mouse is used to double click the map.
maptypechanged	Occurs when the map type changes.
mousedown	Occurs when the left mouse button is pressed when the mouse cursor is over the map.
mousemove	Occurs when the mouse cursor moves over the map.
mouseout	Occurs when the mouse cursor moves out of the area covered by the map.
mouseup	Occurs when the left mouse button is lifted up when the mouse cursor is over the map.
mousewheel	Occurs when the mouse wheel is used when the mouse cursor is over the map.
rightclick	Occurs when the right mouse button is used to click the map.
viewchange	Occurs for every frame of a map view change.
viewchangeend	Occurs when the map view is done changing. This event occurs when a view is the same for one frame of a map view change. For example, if the mouse is used to drag the map to change the view, but pauses during the drag (without releasing the mouse button), viewchangeend occurs twice. You can use the addThrottledHandler method to customize the number of events that occur.
viewchangestart	Occurs when the map view starts changing.

To test out the **viewchange** event open up the **default.html** file and add the following HTML after the map **div**.

```
<span id="coordinateDisplay"></span>
```

To add some style to this **span** tag open the **default.css** file and add the following CSS.

```
#coordinateDisplay {
  position: absolute;
  bottom: 0px;
  left: 50%;
  height: 30px;
  width: 150px;
  margin-left: -75px;
  padding: 5px;
  text-align: center;
  background-color: black;
}
```

Next, open the **default.js** file and add an event handler to the map's **viewchanged** event by adding the following code in the **GetMap** function method after the code for initializing the map.

```
Microsoft.Maps.Events.addHandler(map, 'viewchange', function (e) {
  var center = map.getCenter();
  document.getElementById('coordinateDisplay').innerText = center.latitude.toFixed(5) +
  ',' + center.longitude.toFixed(5);
});
```

As you pan the map you will notice that the coordinate information for the center of the map will appear at the bottom of the screen.



Tip: Some of the events, such as the mouse related events, pass an event argument to the event handler. This event argument can contain important information such as a reference the target object the event is attached to, or the pixel coordinates the mouse event occurred. If you want to get the coordinates of a mouse event, take **pageX** and **pageY** properties of the event argument and pas these through the maps **tryPixelToLocation** function method.

Pushpins, Polylines, Polygons and EntityCollections all have a number of different events that you can make use of. At the time of writing this book there is a bug which causes issues with mouse/tap events on pushpins, polylines and polygons. Having looked into this there is an easy solution. Add the following CSS styles to your app, either in the header of the page or in a CSS file.

```
<style>
  /* Bug fix for mouse events on Polylines and Polygons in IE11 */
  .MicrosoftMapDrawing, .MapPushpinBase {
    pointer-events: auto !important;
  }
</style>
```

Displaying a User's Location

As we saw in Chapter 2 there are two methods within Windows 8 to get the users location; the W3C Geolocation API and the **Windows.Devices.Geolocation.Geolocator** class. The Bing Maps API introduces a third method through the **GeoLocationProvider** class. The benefit of using this class is that it really simplifies integration with the Location services and will also generate the accuracy circle on the map for us. Here is a list of the function methods that are available in the **GeoLocationProvider** class.

Name	Description
addAccuracyCircle	Renders a geo location accuracy circle on the map. The accuracy circle is created with the center at the specified location, using the given radius in meters, and with the specified number of segments for the accuracy circle polygon. Additional options are also available to adjust the style of the polygon.
cancelRequest	Cancels the processing of the current getCurrentPosition request. This method prevents the response from being processed.
getCurrentPosition	Obtains the user's current location and displays it on the map. The accuracy of the user location obtained using this method varies widely depending on the desktop browser or mobile device of the requesting client. Desktop users may experience low user location accuracy (accuracy circles with large radii's), while mobile user location accuracy may be much greater (a few meters).
removeAccuracyCircle	Removes the current geo location accuracy circle.

To add the functionality to display the users location update the **GpsBtn_Tapped** event handler with the following code. This code clears the map and creates an instance of the **GeoLocationProvider** if one doesn't already exist. It then gets the users current position and if successful displays a pushpin and the accuracy circle at the user's location.

```
var geoLocationProvider;

function GpsBtn_Tapped() {
    map.entities.clear();

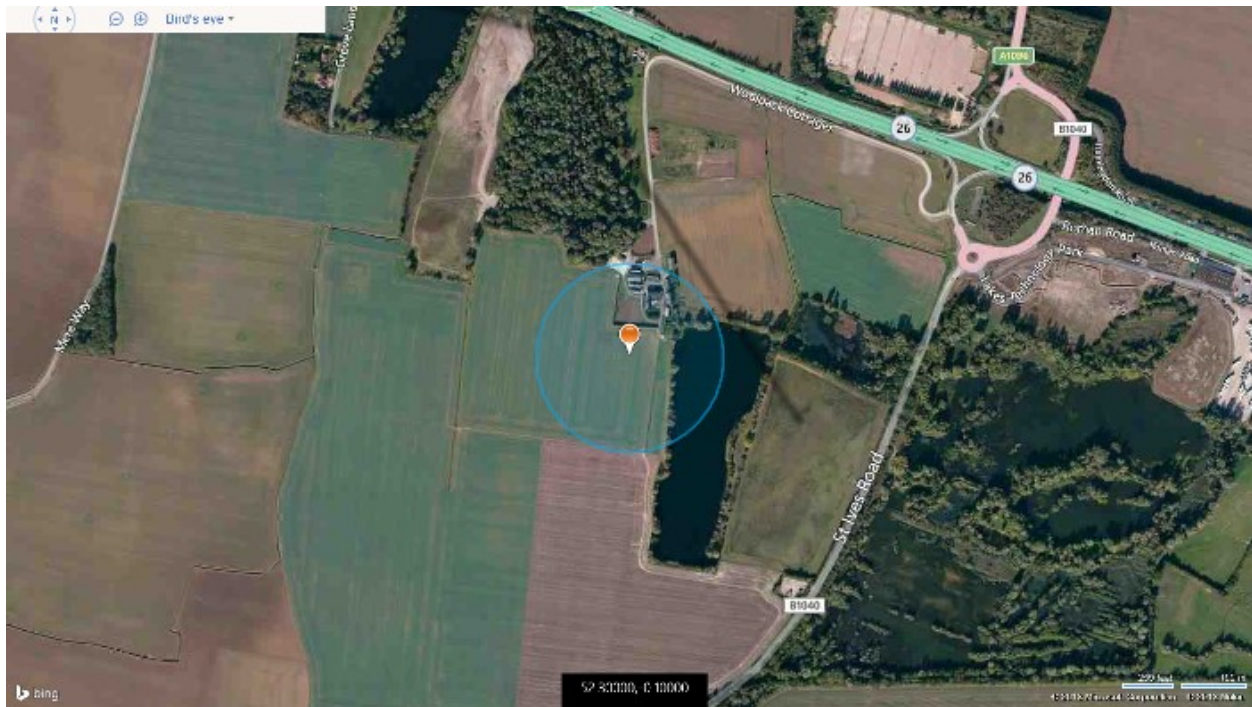
    // Initialize the location provider
    if (!geoLocationProvider) {
        geoLocationProvider = new Microsoft.Maps.GeoLocationProvider(map);
    }

    //Get the users current position
    geoLocationProvider.getCurrentPosition({
        successCallback: function (e) {
            map.entities.push(new Microsoft.Maps.Pushpin(e.center));
        },
        errorCallback: function (e) {
            var dialog = new Windows.UI.Popups.MessageDialog('Unable to locate you.',
"GPS");
            dialog.showAsync();
        }
    });
}
```

Before we get too far ahead of ourselves, we need to add the Location services to the capabilities in the package manifest; otherwise, we won't be able to get any location information from the user's device. To do this, open the **package.appxmanifest** file and select the **Capabilities** tab at the top. Under the **Capabilities** section, check the Location option.

Application	Visual Assets	Capabilities	Declarations	Content URLs	Packaging
Use this page to specify system features or devices that your app can use.					
Capabilities: <ul style="list-style-type: none"> <input type="checkbox"/> Enterprise Authentication <input checked="" type="checkbox"/> Internet (Client) <input type="checkbox"/> Internet (Client & Server) <input checked="" type="checkbox"/> Location <input type="checkbox"/> Microphone <input type="checkbox"/> Music Library <input type="checkbox"/> Pictures Library <input type="checkbox"/> Private Networks (Client & Server) <input type="checkbox"/> Proximity <input type="checkbox"/> Removable Storage <input type="checkbox"/> Shared User Certificates <input type="checkbox"/> Videos Library <input type="checkbox"/> Webcam 		Description: This capability is subject to Store policy. See "More Information" for details. Provides the capability to connect to enterprise intranet resources that require domain credentials. This capability is not typically needed for most apps. More information			

If you run the application and press the GPS button the map will zoom into your location and display a pushpin and circle of accuracy on it.



Creating a Custom Module

Earlier in this chapter you were introduced with the modular framework used by Bing Maps. In this section we are going to create a simply module which we can easily reuse between Bing Maps Window Store and web based apps.

The first task is to determine what type of module you want to create. One thing you may notice if you use both the JavaScript and Native map controls is that the default pushpins are different. Here is a side by side comparison:



In addition to this, in the native map control the color of the pushpin can be easily changed by setting the background color property. The module we are going to create will create a **CirclePushpin** class that looks just like the default pushpin in the Native map control and has the ability to have its color changed. To create this module add a new JavaScript file to the **js** folder of the project called **CirclePushpinModule.js** and add the following code to it.

```
var CirclePushpin = function (loc, options) {
    var text = '', color = new Microsoft.Maps.Color(255, 67, 151, 254);

    //Get text or background color from options
    if (options != null) {
        text = options.text;

        if (options.background) {
            color = options.background;
        }
    } else {
        options = {};
    }

    //Set default options
    options.width = 20;
    options.height = 20;
    options.anchor = new Microsoft.Maps.Point(10, 10);

    //HTML Template for creating the circle pushpin
    var pinTemplate = "<div style='width:20px;height:20px;border:3px solid white;border-
radius:15px;background-color:{color};font:bold 10pt Arial, Helvetica, Sans-Serif;line-
height:20px;text-align:center;color:white;'>{text}</div>";

    //A base Microsoft.Maps.Pushpin
    var basePushpin = new Microsoft.Maps.Pushpin(loc, options);

    //An internal function that applies the HTML template to the pushpin
    function updateLayout() {
        basePushpin.setOptions({
            htmlContent: pinTemplate.replace(/{\color}/gi, color.toHex()).replace(/{\text}/gi,
(!text)? '' : text)
        });
    }

    updateLayout();

    //A function is added to the pushpin to allow the user to set the background color
    basePushpin.setBackground = function (c) {
        color = c;
        updateLayout();
    };

    //A function is added to the pushpin to allow the user to get the background color
    basePushpin.getBackground = function () {
        return color;
    };

    //An event handler is used to detect if the user changed the options of the pushpin.
```



```

Microsoft.Maps.Events.addHandler(basePushpin, 'entitychanged', function () {
    var txt = basePushpin.getText();
    if (text != txt) {
        text = txt;
        updateLayout();
    }
});

return basePushpin;
};

Microsoft.Maps.moduleLoaded('CirclePushpinModule');

```

This module creates a new class called **CirclePushpin**. This class takes in a location and an optional object containing pushpin options, just like the **Pushpin** class. This class creates a custom HTML pushpin that generates a circle by using rounded borders. This class also adds the following function methods to the pushpin to make it more customizable.

Name	Type	Description
getBackgorund	Microsoft.Maps.Color	Gets the background color of the circle pushpin.
setBackground	Microsoft.Maps.Color	Sets the background color of the circle pushpin.

To add the functionality to use this custom module first open the **default.html** file and register the module by adding the following HTML in the head of the page after the script reference for the **BingMapsIntro.js** file.

```

<!-- Register code for the Custom Module -->
<script>Microsoft.Maps.registerModule('CirclePushpinModule');</script>
<script type="text/javascript" src="/js/CirclePushpinModule.js"></script>

```

Next open the **BingMapsIntro.js** file and update the **CustomModuleBtn_Tapped** event handler with the following code. This code clears the map and loads the custom circle pushpin module if it isn't already loaded. It then creates a **CirclePushpin** at the center of the map and adds the text "1" to it.

```

function CustomModuleBtn_Tapped() {
    map.entities.clear();

    if (CirclePushpin) {
        var center = map.getCenter();

        //Create a Circle Pushpin
        var pin = new CirclePushpin(center, { text: '1' });
        map.entities.push(pin);
    } else {
        //Load the module if it isn't already loaded.
        Microsoft.Maps.loadModule('CirclePushpinModule', {
            callback: function () {
                CustomModuleBtn_Tapped();
            }
        });
    }
}

```

If you press the custom module button you will see the custom circle pushpin appear at the center of the map.



Chapter Summary

In this chapter you were introduced to JavaScript Bing Maps Windows Store. Here is a short summary of some key points from this chapter.

- Full documentation on the Bing Maps JavaScript API can be found here: <http://bit.ly/18zXRwQ>. You can also try out the Bing Maps Interactive SDK which is a great place to try out things in the JavaScript API quickly in a web browser: <http://bit.ly/1bKDFuD>.
- The Bing Maps JavaScript API makes use of a modular framework which allows you to load in the features and functionalities your app needs when it needs them. In addition to the modules available through the Bing Maps API, there are a large number of community created modules available through the Bing Maps V7 Modules CodePlex project (<http://bit.ly/1buEQ19>).
- The Bing Maps control has aerial, Birdseye, Collins Bartholomew, Ordnance Survey and road maps.
- You can localize the map by setting the **culture** and **homeRegion** properties when loading the **Microsoft.Maps.Map** module. Bing Maps supports 116 languages. A full list of supported cultures can be found here: <http://bit.ly/1aY3gDh>.
- **EntityCollections** can be used to layer data on the map.
- You can add **Pushpins**, **Polylines**, **Polygons**, **TileLayers** and **EntityCollections** to the map.
- The Advance Shapes module can be used to add support for polygons with holes.
- It is a best practice to separate your data and infoboxes by using two layers. By doing this you will be able to ensure that your infobox is always rendered above the pushpins on the map.
- When you have a lot of pushpins and only want to show one infobox at a time the best approach is to create one Infobox and to reuse it rather than creating an infobox for each pushpin. By doing this you will greatly reduce the number of DOM elements created by the application which will provide better performance. Also, separate your data from your infoboxes using layers to ensure your infoboxes always show up above your data. Take a look at this blog post for more information: <http://bit.ly/1daUGDd>.
- Tile layers are an excellent way to visualize large data sets on a map. Often the size of an image of the data is much smaller than the data itself. The smaller size means faster downloads and less objects for the map to render which makes for a better performance. Some good sources for tile layers include:
 - OpenStreetMap derived services - <http://bit.ly/1aCamOA>
 - Data.gov - <http://1.usa.gov/15jGq5C>

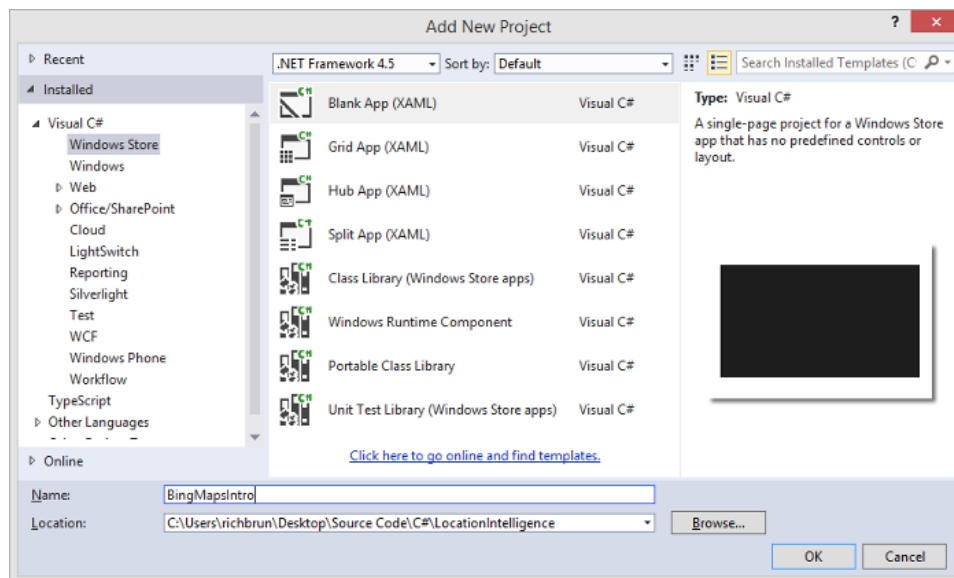
- USGS - <http://bit.ly/15RbIDA>
 - Geoserver - <http://bit.ly/14LnTve>
- The Search module provides geocoding, reverse geocoding, and point of interest search functionalities. Geocoding takes an address and determines its location on the map. Reverse geocoding finds the closest known location to a coordinate.
- The Directions module provides the tools needed for calculating routes between locations for driving, walking and public transport. It also provides support for updating the route by dragging it and for generating nicely formatted instructions.
- The Traffic module provides traffic flow and traffic incident data. A list of countries where traffic data is available can be found here: <http://bit.ly/130ytc>
- The Venue Map module provides the tools needed to find and load Venue Maps. Venue Maps are often used to show indoor floor plans of complex buildings such as malls or airports.

Chapter 4: Bing Maps Native API

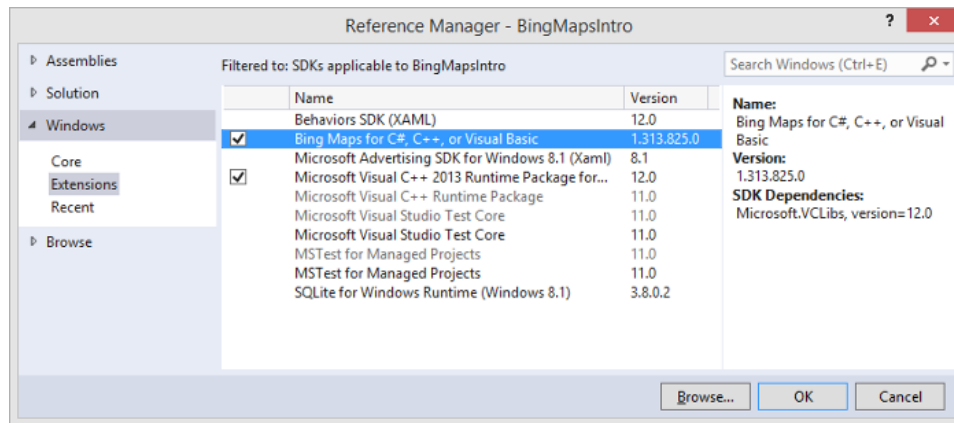
The Bing Maps Native API for Windows Store apps provides you with the greatest user experience and performance available by any mapping API in Windows 8. This API has been developed using C++ so that it can take full advantage of things such as hardware acceleration. Before you can make use of the Bing Maps SDK in a Windows Store application you have to first make sure you have it installed (<http://bit.ly/11TLdei>). You will also need a Bing Maps account and key as discussed in Chapter 1.

Adding Bing Maps to a Windows Store app

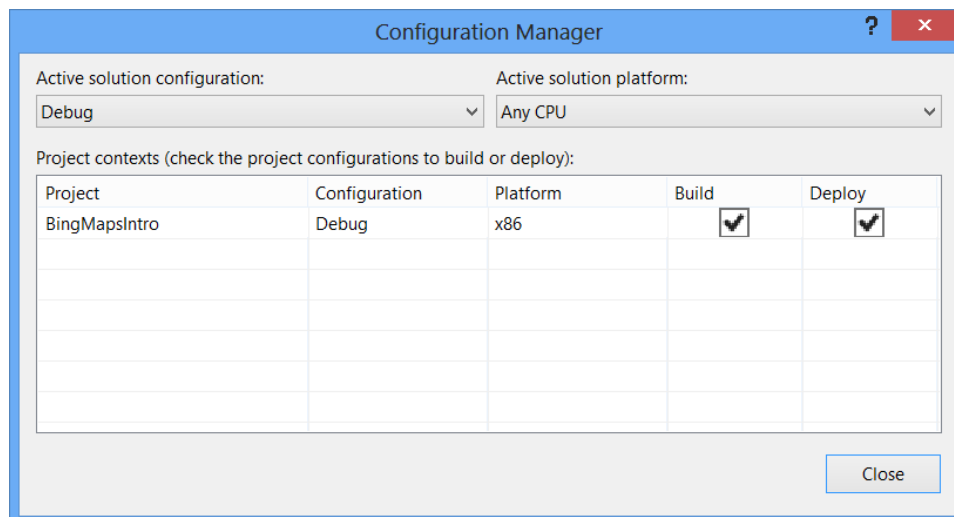
Open up Visual Studios and create a new project. In the window that opens, select **Visual C# (or Visual Basic)** -> **Windows Store**. Select the Blank App template, and call the application **BingMapsIntro** and press OK.



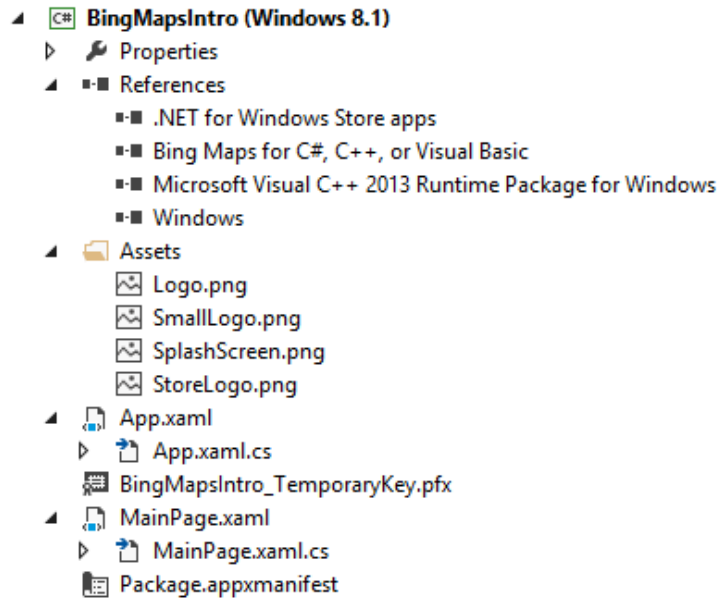
Next add a reference to the Bing Maps SDK. To do this right click on the **References** folder and press **Add Reference**. Select **Windows -> Extensions**, and then select **Bing Maps for C#, C++ and Visual Basic**. If you do not see this option, be sure to verify that you have installed the Bing Maps SDK for Windows Store apps. While you are here, also add a reference to the **Microsoft Visual C++ Runtime Package** as this is required by the Bing Maps SDK when developing using C# or Visual Basic.



You may notice a little yellow indicator on the references that you just added. The Microsoft Visual C++ Runtime Package requires that the **Active solution platform** in the Visual Studio project be set to one of the following options: **ARM**, **x86** or **x64**. The reason for this is that C++ libraries need to be compiled against each processor architecture separately as it can't be compiled such that it works across all CPU types. To set the **Active solution platform** in Visual Studios right click on the Solution folder and select Properties. Then, go to **Configuration Properties -> Configuration**. Locate your project and under the **Platform** column and set the target platform to x86. Press OK and the yellow indicator will disappear from your references.



At this point your application will look something like this in the solution folder.



Now you can add a map to your application. To do this, open the **MainPage.xaml** file and add the Bing Maps SDK as a namespace, `xmlns:m="using:Bing.Maps"`, at the top of the file. Next add a **Map** object to the **Grid** control and add your Bing Maps key to the credentials properties of the map. Give the map a name of **MyMap**. When you are done your XAML will look like this:

```
<Page
  x:Class="BingMapsIntro.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:BingMapsIntro"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:m="using:Bing.Maps"
  mc:Ignorable="d">

  <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <m:Map Name="MyMap" Credentials="YOUR_BING_MAPS_KEY"/>
  </Grid>
</Page>
```

You can now build and run the application (F5) you will see your app open up with a full screen map of the world.



Next open up the **MainPage.xaml** file and add the following XAML after the **Grid** control. This will add several buttons to the bottom app bar of the application which we will use to test out different features of the API. When the application is running you will simply need to swipe up from the bottom of the app to display these buttons.

```
<Page.BottomAppBar>
    <AppBar>
        <StackPanel Orientation="Horizontal">
            <AppBarButton Label="Add Pushpin" Tapped="AddPushpin_Tapped">
                <AppBarButton.Icon>
                    <FontIcon Glyph="⬮"/>
                </AppBarButton.Icon>
            </AppBarButton>

            <AppBarButton Label="Add Polyline" Tapped="AddPolyline_Tapped">
                <AppBarButton.Icon>
                    <FontIcon Glyph="⬭"/>
                </AppBarButton.Icon>
            </AppBarButton>

            <AppBarButton Label="Add Polygon" Tapped="AddPolygon_Tapped">
                <AppBarButton.Icon>
                    <FontIcon Glyph="⬮"/>
                </AppBarButton.Icon>
            </AppBarButton>

            <AppBarButton Label="Add Tile Layer" Tapped="AddTileLayer_Tapped">
                <AppBarButton.Icon>
                    <FontIcon Glyph="⬮"/>
                </AppBarButton.Icon>
            </AppBarButton>

            <AppBarButton Label="Toggle Traffic" Tapped="ToggleTraffic_Tapped">
```

```

        <AppBarButton.Icon>
            <FontIcon Glyph="&#xE0C3;"/>
        </AppBarButton.Icon>
    </AppBarButton>

    <AppBarButton Label="Add User Control" Tapped="AddUserControl_Tapped">
        <AppBarButton.Icon>
            <FontIcon Glyph="&#xE187;"/>
        </AppBarButton.Icon>
    </AppBarButton>

    <AppBarButton Label="Add Infobox" Tapped="AddInfobox_Tapped">
        <AppBarButton.Icon>
            <FontIcon Glyph="&#xE134;"/>
        </AppBarButton.Icon>
    </AppBarButton>

    <AppBarButton Label="Geocode" Tapped="GeocodeBtn_Tapped" >
        <AppBarButton.Icon>
            <FontIcon Glyph="&#xE11A;"/>
        </AppBarButton.Icon>
    </AppBarButton>

    <AppBarButton Label="Reverse Geocode" Tapped="ReverseGeocodeBtn_Tapped">
        <AppBarButton.Icon>
            <FontIcon Glyph="&#xE128;"/>
        </AppBarButton.Icon>
    </AppBarButton>

    <AppBarButton Label="Directions" Tapped="DirectionsBtn_Tapped">
        <AppBarButton.Icon>
            <FontIcon Glyph="&#xE1D1;"/>
        </AppBarButton.Icon>
    </AppBarButton>

    <AppBarButton Label="Load Venue Map" Tapped="LoadVenueMap_Tapped">
        <AppBarButton.Icon>
            <FontIcon Glyph="&#xE0C3;"/>
        </AppBarButton.Icon>
    </AppBarButton>
</StackPanel>
</AppBar>
</Page.BottomAppBar>

```

Now open the **MainPage.xaml.cs** file and add the following button event handlers. We will add the functionality to these event handlers later.

C#

```

private void AddPushpin_Tapped(object sender, Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
}

private void AddPolyline_Tapped(object sender, Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
}

private void AddPolygon_Tapped(object sender, Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
}

```



```

}

private void AddTileLayer_Tapped(object sender, Windows.UI.Xaml.Input.TappedRoutedEventArgs
e)
{
}

private async void ToggleTraffic_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
}

private void AddUserControl_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
}

private void AddInfobox_Tapped(object sender, Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
}

private async void GeocodeBtn_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
}

private async void ReverseGeocodeBtn_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
}

private async void DirectionsBtn_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
}

private async void LoadVenueMap_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
}

```

Visual Basic

```

Private Sub AddPushpin_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
End Sub

Private Sub AddPolyline_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
End Sub

Private Sub AddPolygon_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
End Sub

Private Sub AddTileLayer_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
End Sub

```

```

Private Async Sub ToggleTraffic_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
End Sub

Private Sub AddUserControl_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
End Sub

Private Sub AddInfobox_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
End Sub

Private Async Sub GeocodeBtn_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
End Sub

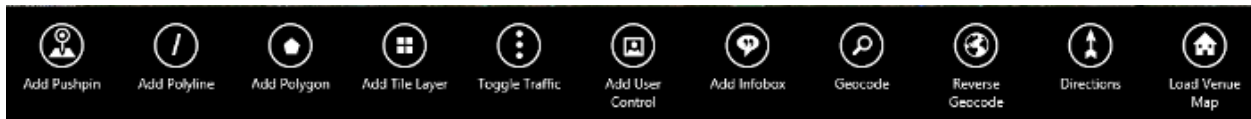
Private Async Sub ReverseGeocodeBtn_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
End Sub

Private Async Sub DirectionsBtn_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
End Sub

Private Sub LoadVenueMap_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
End Sub

```

If you run the application and swipe up from the bottom or right click you will see a black panel with all the buttons on it appear. Pressing any of the buttons won't do anything as we haven't added any logic to the event handlers yet.



Customizing the Map

You can customize how the map loads using a number of different options on the map. These options can be specified in the XAML or from code and can be programmatically changed at any time. There are a large number of options that you can modify. Not all properties support data binding. The following table is a list of properties that you can modify to customize the map.

Name	Type	Supports Data Binding	Description
Center	Location	No	Gets or sets the location of the center of the map.
CopyrightPosition	MapForegroundPosition	No	Gets or sets the position of the map copyright string.
Credentials	String	Yes	The map credentials used to authenticate the map.
Culture	String	Yes	The culture code used to set the language of the map.
Heading	double	No	The directional heading of the map. The heading is represented in geometric degrees with 0 or 360 = North, 90 = East, 180 = South, and 270 = West. You can set the heading by using the SetHeading method.

HomeRegion	String	Yes	The home region of the map. Used in conjunction with the culture code.
LogoPosition	MapForegroundPosition	Yes	The location of the Bing Maps logo.
MapType	MapType	No	The map imagery type, or style being displayed.
ScaleBarPosition	MapForegroundPosition	Yes	The location of the scale bar.
ShowBreadcrumb	bool	Yes	A bool indicating whether to show a breadcrumb control. The breadcrumb control shows the current map center location's geography hierarchy. For example, if the map is centered over Seattle, the breadcrumb control displays "World, United States, WA". The default value is false.
ShowBuildings	bool	Yes	A bool indicating whether to display building footprints. The default is true.
ShowNavigationBar	bool	Yes	A bool indicating whether to display map navigation controls. The default is true.
ShowScaleBar	bool	Yes	A bool indicating whether the scale bar is visible. The default is true.
ShowTraffic	bool	Yes	A bool indicating whether to display traffic on the map. The default is false.
SuppressNetworkRequests	bool	Yes	A bool indicating whether to suppress network requests. The default is false.
ViewRestriction	MapViewRestriction	No	The restrictions for changing the map view. The default value is <code>ZoomOutToWholeWorld</code> .
ZoomLevel	double	No	The zoom level of the map.

Most of these properties make use of simple data types such as double, string and Boolean while others use enumerators or the Location class which is used to represent coordinates.


MapForegroundPosition Enumerator




This enumerator contains predefined locations on the map which are used for positioning various UI elements on the map.

Name	Description
BottomLeft	The bottom-left corner of the map.
BottomRight	The bottom-right corner of the map.
TopLeft	The top-left corner of the map.
TopRight	The top-right corner of the map.

MapType Enumerator

This enumerator is used to specify the type of map style that should be displayed by the map.

Name	Description	Example
Aerial	The map displays aerial imagery.	

Birdseye	The map displays an angled view of aerial imagery.	
Empty	The map does not display any imagery. Use this option if you want to display custom imagery instead of Bing Maps imagery.	
HighContrast	The map displays high contrast road imagery.	
Road	The map displays road imagery.	

MapViewRestriction Enumerator

When the map is zoomed out to a high level, it is possible to see more than one instance of the world map unless you restrict the view to one world map. This enumerator allows you to specify how the map should render when more than one instance of the map could be displayed in the current view.

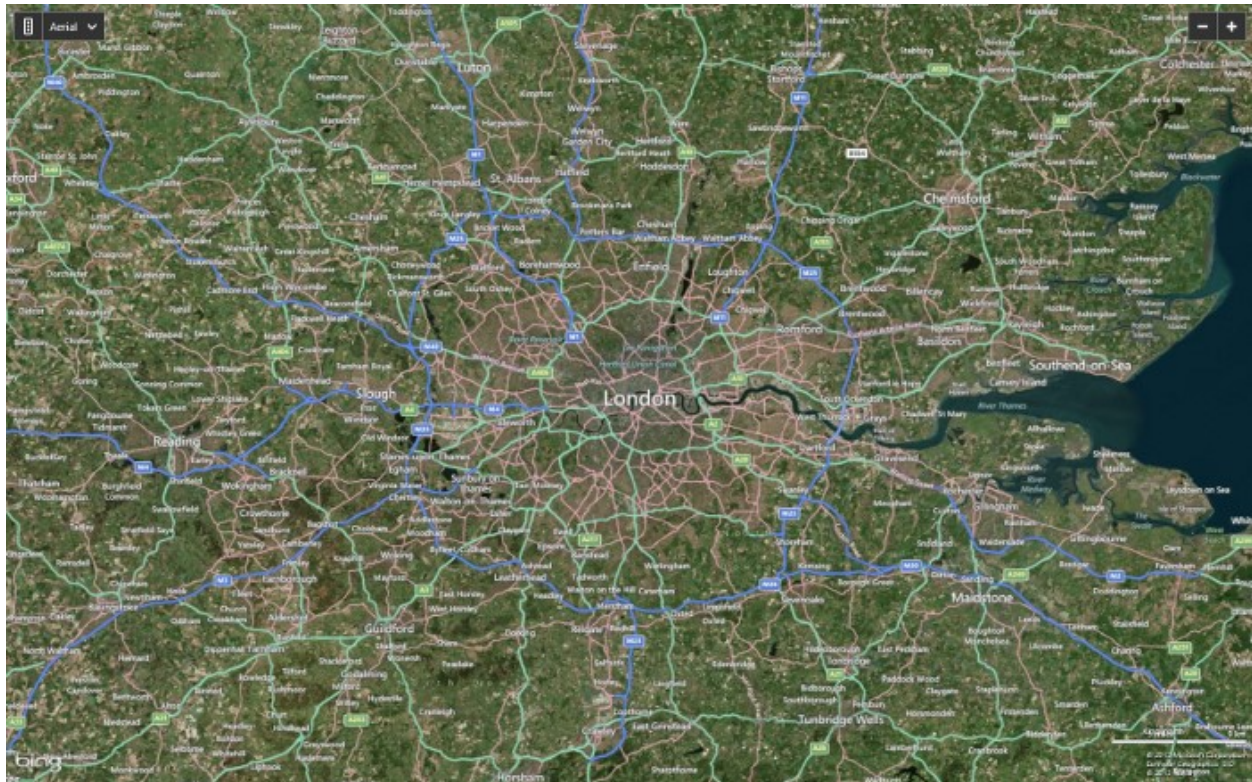
Name	Description
MapOnly	The map view can display multiple world maps.
None	The map view has no restrictions.
OneWorldOnly	The map view can only display one world map.
ZoomOutToWholeWorld	The map view can be zoomed out to display one or more world maps.

Customizing the Map when Loading

As mentioned before you can set the map properties in XAML or in code. The following is an example of how to use XAML to set the map type to aerial, the zoom level to 10, and center the map on London, UK (51.50632, -0.12714).

```
<m:Map Name="MyMap" Credentials="YOUR_BING_MAPS_KEY"
    MapType="Aerial" ZoomLevel="10">
    <m:Map.Center>
        <m:Location Latitude="51.50632" Longitude="-0.12714" />
    </m:Map.Center>
</m:Map>
```

This results in the map being loaded, zoomed in over London, UK with the aerial imagery being displayed.



The following code can be used to do the same thing programmatically in the **MainPage.xaml.cs** file.

C#

```
using Bing.Maps;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;

namespace BingMapsIntro
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();

            MyMap.MapType = MapType.Aerial;
            MyMap.ZoomLevel = 10;
            MyMap.Center = new Location(51.50632, -0.12714);
        }
    }
}
```

Visual Basic

```
Imports Bing.Maps

Public NotInheritable Class MainPage
    Inherits Page
```



```

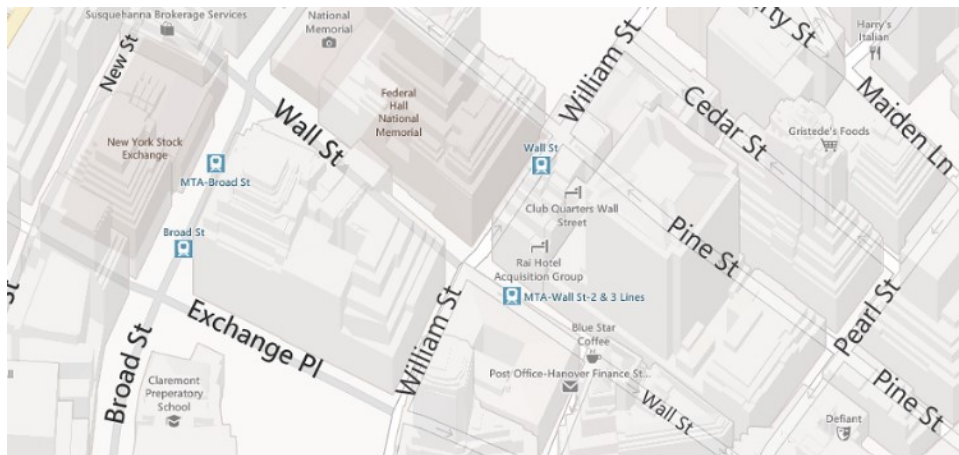
Public Sub New()
    InitializeComponent()

    MyMap.MapType = MapType.Aerial
    MyMap.ZoomLevel = 10
    MyMap.Center = New Location(51.50632, -0.12714)
End Sub
End Class

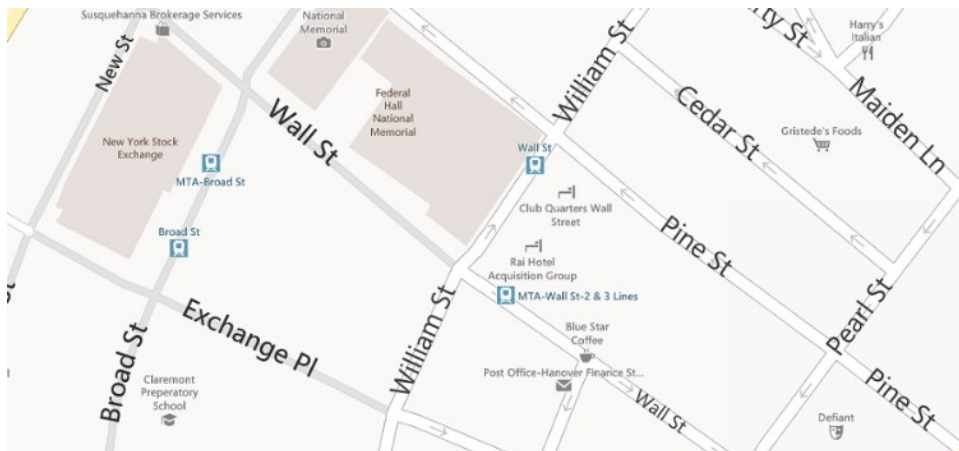
```

Building Footprints and a Breadcrumb Control

There are a number of features of the map that can be turned on or off. Two of these features are building footprints and a breadcrumb control. The building footprints are on by default and are displayed when zoomed into the map while viewing the Road imagery. The building footprints look like the following.



Disabling the building footprint of previous map will result in the following map.



The breadcrumb control provides a hierarchy of locations based on where the map is centered over. This hierarchy usually takes the form of World -> Country -> Admin District -> City. Besides being a nice way to keep track of what you are looking at it also provides a useful way for jumping up a level to see a higher level view of an area. Similarly to building footprints you can show or hide this control by setting the **ShowBreadcrumb** property of the map to true or false. By default the breadcrumb is disabled. Here is an example of what the breadcrumb looks like.

World > United Kingdom > England > London

Culture and Localization

Bing Maps supports 116 languages. By default the map control will automatically render in the language that the user's device is set to. This makes it easy for you as a developer as you don't need to worry about setting the language of the map. However, if you for some reason you would like to limit your application to one language you can force the map to use a specific language by specifying the **Culture** property on the map. A full list of supported cultures can be found here: <http://bit.ly/1aY3gDh>. The map culture can be set using XAML or code as shown below.

XAML

```
<m:Map Name="MyMap" Culture="en-US"/>
```

C#

```
MyMap.Culture = "en-US";
```

Visual Basic

```
MyMap.Culture = "en-US"
```

In addition to the **Culture** property of the map there is also a **HomeRegion** property. This property also defaults to the settings on the user's device but can be hard coded if desired. The home region property is used to determine how to handle geopolitical issues on the map. For instance, many countries recognize the Persian Gulf as a body of water in the Middle East, however many Arabic countries recognize this same body of water as the Arabian Gulf. As such the home region property allows the map to decide which name to use. You can hard code the home region by passing in a 2 letter country code as a string value. Not all regions are currently supported. If you come across a region that isn't supported the map will be empty or have an icon indicating there is no maps available. To get around this you can simply hard code in a value that you feel comfortable using for that region. For instance, at the time of writing this book India which has a country code of IN is not yet support. In this case you could enable maps on devices that have a home region of IN by hard coding this property to US. Like the culture property, the home region can also be set using XAML and code as shown below.

XAML

```
<m:Map Name="MyMap" HomeRegion="US"/>
```

C#

```
MyMap.HomeRegion = "US";
```

Visual Basic

```
MyMap.HomeRegion = "US"
```

Changing the Map View

There are a number of different ways to change the map view. One way is to simply modify one of the view related properties on the map such as **Center**, **ZoomLevel**, **Heading**, or **MapType**. There are also a number of methods on the map that could be used. These methods not only allow you to pass in a value to change the map view but also a

timespan for how long the animation to the new map view should take. Here is a list of the methods you can use to modify the map view:

Method	Description
SetHeading	Sets the directional heading of the map. The heading is represented in geometric degrees with 0 or 360 = North, 90 = East, 180 = South, and 270 = West. Make sure the RotateEnabled property is set to true before trying to change the map heading.
SetView	Sets the view of the map using multiple view properties.
SetZoomLevel	Sets the zoom level of the map view and the animation time.
SetZoomLevelAroundPoint	Sets the zoom level of the map view to the specified value while maintaining the screen position of the given point.

The **SetView** method gives you the ability to use multiple view properties in a single method to change the view. In addition to this you can also pass in a **LocationRect** object as a bounding box which will cause the map to zoom and center on the map such that it is positioned to show the full bounding box. Note that this doesn't mean the corners of the map will align with the bounding box as the maps width/height ratio will likely be different from that of the bounding box.

Overlaying Content on the Map

Maps have evolved from being a source of information to being a canvas that information is displayed on top of. This information can take many forms and can be represented in several different ways. The following sections show how to add a number of different types of information to the map using both XAML and code.

Layering Content

You can add content to Bing Maps in a number of different ways. There are two different types of layers available in the native Bing Maps API using the **MapLayer** and **MapShapeLayer** classes. The **MapLayer** class allows you to add Pushpins, and **UIElement**'s to it. In addition to that you can also nest other **MapLayer** objects in side of it. The **MapLayer** class provides the following methods.

Name	Description
GetPosition	Returns the position of a Pushpin or UIElement .
GetPositionAnchor	Returns the position anchor of a Pushpin or UIElement .
SetPosition	Sets the position of a Pushpin or UIElement added to a MapLayer .
SetPositionAnchor	Sets the position anchor of a Pushpin or UIElement . The position anchor defaults to (0,0) which means the element will have its top-left position aligned with the position value. For a round pushpin this should be set to (Width / 2, Height / 2) to center the element over its position.

The **Children** property on the map control derives from the **MapLayer** class and is how you add pushpins and custom **UIElement**'s to the map. The following shows how to add a **MapLayer** as a child of a map.

```
<m:Map Name="MyMap">
  <m:Map.Children>
    <m:MapLayer Name="DataLayer"/>
  </m:Map.Children>
</m:Map>
```

MapLayers can also be added to the map through code as shown below.

C#

```
MapLayer DataLayer = new MapLayer();
MyMap.Children.Add(DataLayer);
```

Visual Basic

```
Dim DataLayer = New MapLayer()
MyMap.Children.Add(DataLayer)
```

Tip: It is a best practice to separate your data and infoboxes by using two layers. By doing this you will be able to ensure that your infobox is always rendered above the pushpins on the map.

The second type of layer is the **ShapeLayer** class. This class is used to add objects that derive from the **MapShape** class such as the **MapPolyline** and **MapPolygon** classes. Shape layers can be defined in XAML however shape layers defined in XAML cannot be accessed programmatically. This means that if you wanted to add any shapes to a shape layer defined in XAML you would also have to define the shapes in XAML as well. For example:

```
<m:Map Name="MyMap">
  <m:Map.ShapeLayers>
    <m:MapShapeLayer>
      <!-- Define Shapes -->
    </m:MapShapeLayer>
  </m:Map.ShapeLayers>
</m:Map>
```

Luckily shape layers can also be added to the map through code and when done this way you can programmatically add shapes to the shape layer. Here is how you would add a shape layer to a map in code.

C#

```
MapShapeLayer PolyLayer = new MapShapeLayer();
MyMap.ShapeLayers.Add(PolyLayer);
```

Visual Basic

```
Dim PolyLayer = New MapShapeLayer()
MyMap.ShapeLayers.Add(PolyLayer)
```

In the **BingMapsIntro** project open up the **MainPage.xaml** file and add two map layer's called **DataLayer** and **InfoboxLayer**. These layers will be used in code samples later in this chapter. The XAML to create these layers looks like this.

```
<m:Map Name="MyMap">
  <m:Map.Children>
    <m:MapLayer Name="DataLayer"/>
    <m:MapLayer Name="InfoboxLayer" Visibility="Collapsed"/>
  </m:Map.Children>
</m:Map>
```

In addition to this open the **MainPage.xaml.cs** file and add a global **MapShapeLayer** to the map and a method for clearing the map and layers such that it looks like the following.

C#

```

using Bing.Maps;
using Bing.Maps.Directions;
using Bing.Maps.Search;
using System;
using Windows.UI.Popups;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

namespace BingMapsIntro
{
    public sealed partial class MainPage : Page
    {
        private MapShapeLayer PolyLayer;

        public MainPage()
        {
            this.InitializeComponent();

            PolyLayer = new MapShapeLayer();
            MyMap.ShapeLayers.Add(PolyLayer);
        }

        private void ClearMap()
        {
            DataLayer.Children.Clear();
            PolyLayer.Shapes.Clear();
            MyMap.TileLayers.Clear();

            InfoboxLayer.Visibility = Visibility.Collapsed;
        }

        //Button Handlers...
    }
}

```

Visual Basic

```

Imports Bing.Maps
Imports Bing.Maps.Search
Imports Bing.Maps.Directions
Imports Windows.UI.Popups

Public NotInheritable Class MainPage
    Inherits Page

    Private PolyLayer As MapShapeLayer

    Public Sub New()
        InitializeComponent()

        PolyLayer = New MapShapeLayer()
        MyMap.ShapeLayers.Add(PolyLayer)
    End Sub

    Private Sub ClearMap()
        DataLayer.Children.Clear()
    End Sub

```

```

        PolyLayer.Shapes.Clear()
        MyMap.TileLayers.Clear()

        InfoboxLayer.Visibility = Visibility.Collapsed
    End Sub

    ''Button Handlers...

End Class

```

Pushpins

Pushpins, sometimes also referred to as markers on other mapping platforms, are one of the primary ways of marking a location on a map. There are a few properties which can be used to customize a pushpin. These properties include:

Name	Type	Description
Background	Color	Gets or sets the background color of the pushpin.
Location	Location	Gets or sets the location of the pushpin.
Text	String	Gets or sets the text for a pushpin.

Pushpins can be added to the map through XAML or code. Here is an example of how you can add a pushpin to the map using XAML.

```

<m:MapLayer Name="DataLayer">
    <m:Pushpin>
        <m:MapLayer.Position>
            <m:Location Latitude="0" Longitude="0"/>
        </m:MapLayer.Position>
    </m:Pushpin>
</m:MapLayer>

```

More often than not you will likely find that you are adding pushpins to the map through code. In the **MainPage.xaml.cs** file add the following code to the **AddPushpin_Tapped** button handler. This code will clear the map and add a new pushpin at the center of the map that has a text label set to 1 and a background color of red.

C#

```

ClearMap();

//Create Pushpin
Pushpin pin = new Pushpin();
pin.Text = "1";
pin.Background = new Windows.UI.Xaml.Media.SolidColorBrush(Windows.UI.Colors.Red);

MapLayer.SetPosition(pin, MyMap.Center);
DataLayer.Children.Add(pin);

```

Visual Basic

```

ClearMap()

'Create Pushpin
Dim pin As New Pushpin()
pin.Text = "1"
pin.Background = New Windows.UI.Xaml.Media.SolidColorBrush(Windows.UI.Colors.Red)

```

```
MapLayer.SetPosition(pin, MyMap.Center)
DataLayer.Children.Add(pin)
```

If you click on this button a pushpin will appear at the center of the map.



Tip: The Pushpin class inherits from the **UIElement** class, as such it has a property called **Tag** which is an **object** type. This property is handy for storing metadata that is related to the pushpin such as a unique identifier, or the data needed to populate an infobox.

Polylines

Polylines, also known as LineStrings, allow you to draw connected lines on a map. You can use the following properties to customize a polyline.

Name	Type	Description
Color	Windows.UI.Color	Gets or sets the color of the polyline.
Locations	LocationCollection	Gets or sets the locations that define the shape.
Width	int	Gets or sets the width of the polyline, in pixels.

Polylines can be added to the map through XAML or code using a **MapShapeLayer** as shown below.

```
<m:Map>
  <m:Map.ShapeLayers>
    <m:MapShapeLayer>
      <m:MapShapeLayer.Shapes>
        <m:MapPolyline Color="Red">
          <m:MapPolyline.Locations>
            <m:Location Latitude="0" Longitude="10" />
            <m:Location Latitude="10" Longitude="10" />
            <m:Location Latitude="10" Longitude="0" />
          </m:MapPolyline.Locations>
        </m:MapPolyline>
      </m:MapShapeLayer.Shapes>
    </m:MapShapeLayer>
  </m:Map.ShapeLayers>
</m:Map>
```

To add polylines programmatically open the **MainPage.xaml.cs** file add the following code to the **AddPolyline_Tapped** button handler. This code will clear the map and add a new polyline based on the center of the map that is red in color and has a width of 10 pixels.

C#

```
ClearMap();

//Generate collection of coordinates
LocationCollection coords = new LocationCollection();
coords.Add(MyMap.Center);
coords.Add(new Location(MyMap.Center.Latitude + 1, MyMap.Center.Longitude + 1));

//Create a polyline
MapPolyline line = new MapPolyline();
line.Locations = coords;
line.Color = Windows.UI.Color.FromArgb(150, 255, 0, 0);
line.Width = 10;

//Add the polyline to the PolyLayer
PolyLayer.Shapes.Add(line);
```

Visual Basic

```
ClearMap()

''Generate collection of coordinates
Dim coords = New LocationCollection()
coords.Add(MyMap.Center)
coords.Add(New Location(MyMap.Center.Latitude + 1, MyMap.Center.Longitude + 1))

''Create a polyline
Dim line = New MapPolyline()
line.Locations = coords
line.Color = Windows.UI.Color.FromArgb(150, 255, 0, 0)
line.Width = 10

''Add the polyline to the PolyLayer
PolyLayer.Shapes.Add(line)
```

If you click on this button a polyline that starts at the center of the map will be displayed.



Polygons

Polygons allow you to draw an area on a map. You can use the following properties to customize a polygon.

Name	Type	Description
FillColor	Windows.UI.Color	Gets or sets the color to use to fill the polygon.
Locations	LocationCollection	Gets or sets the locations that define the shape.

Polylines can be added to the map through XAML or code using a **MapShapeLayer** as shown below.

```
<m:Map>
  <m:Map.ShapeLayers>
    <m:MapShapeLayer>
      <m:MapShapeLayer.Shapes>
        <m:MapPolygon FillColor="Red">
          <m:MapPolygon.Locations>
            <m:Location Latitude="0" Longitude="10" />
            <m:Location Latitude="10" Longitude="10" />
            <m:Location Latitude="10" Longitude="0" />
          </m:MapPolygon.Locations>
        </m:MapPolygon>
      </m:MapShapeLayer.Shapes>
    </m:MapShapeLayer>
  </m:Map.ShapeLayers>
</m:Map>
```

To add polylines programmatically open the **MainPage.xaml.cs** file add the following code to the **AddPolygon_Tapped** button handler. This code will clear the map and add a new polygon based on the center of the map that is green in color.

C#

```
ClearMap();

//Generate collection of coordinates
LocationCollection coords = new LocationCollection();
coords.Add(MyMap.Center);
coords.Add(new Location(MyMap.Center.Latitude + 1, MyMap.Center.Longitude + 1));
coords.Add(new Location(MyMap.Center.Latitude - 1, MyMap.Center.Longitude + 1));

//Create a polygon
```



```

MapPolygon polygon = new MapPolygon();
polygon.Locations = coords;
polygon.FillColor = Windows.UI.Color.FromArgb(150, 0, 255, 0);

//Add the polygon to the PolyLayer
PolyLayer.Shapes.Add(polygon);

```

Visual Basic

ClearMap()

```

'Generate collection of coordinates
Dim coords = New LocationCollection()
coords.Add(MyMap.Center)
coords.Add(New Location(MyMap.Center.Latitude + 1, MyMap.Center.Longitude + 1))
coords.Add(New Location(MyMap.Center.Latitude - 1, MyMap.Center.Longitude + 1))

'Create a polygon
Dim polygon = New MapPolygon()
polygon.Locations = coords
polygon.FillColor = Windows.UI.Color.FromArgb(150, 0, 255, 0)

'Add the polygon to the PolyLayer
PolyLayer.Shapes.Add(polygon)

```

If you click on this button a polygon in the shape of a triangle will appear on the map.



Tile Layers

As we saw in Chapter 1, Bing Maps makes use of a tiling system to display maps of varying levels of details. It is possible to overlay custom tile layers on top of the map control. These layers can be used to replace the base maps completely, or overlaid on top the base maps to add an additional dimension to the data.

Tile layers are an excellent way to visualize large data sets on a map. There are two benefits to this approach. The first is that an image of the data is likely much smaller than the raw data itself, this means the map will be able to download the data faster. The second is that there will only ever be a few dozen tiles loaded on the map at the same time, the less objects the map control has to reposition when panning and zoom the better performance and the better the overall user experience will be.

The **MapTileLayer** class is used to define a tile layer in Bing Maps. Tile layers can be added to the map through the **TileLayers** property on the map. The **MapTileLayer** class has the following properties which can be used to create a tile layer.

Name	Type	Description
Bounds	LocationRectCollection	Defines the areas where the tile layer is available. A null value (default) displays the tile layer everywhere.
Opacity	double	Gets or sets the opacity of the tile layer, defined by a double between 0 (not visible) and 1.
FillMissingTiles	bool	Specifies if tiles that are not available at the requested zoom level are fill in using tiles from the nearest zoom level available.
TileSource	string	Used to define the URL source of the tiles. Do not use this property if you are using the GetTileUri event and callback function.
MinZoomLevel, MaxZoomLevel	double	Defines the minimum and maximum zoom level to display in the tile layer. The default values are 1 and 20.
Visible	bool	Used to specify if the tile layer is shown.
ZIndex	int	Defines the z-index of the tile layer with respect to other items on the map.

The **MapTileLayer** class also has the following events available.

Name	Description
GetTileUri	Occurs when a map tile is requested and returns the map tile URL.
TileDownloadCompleted	Occurs when all map tiles for the current view are downloaded.

The simplest implementation of the **MapTileLayer** class is to set the **TileSource** to a URL which uses a quadkey value to access map tiles as shown here.

C#

```
MapTileLayer tileLayer = new MapTileLayer();
tileLayer.TileSource = "http://example.com/{quadkey}.png";
MyMap.TileLayers.Add(tileLayer);
```

Visual Basic

```
Dim tileLayer = New MapTileLayer()
tileLayer.TileSource = "http://example.com/{quadkey}.png"
MyMap.TileLayers.Add(tileLayer)
```

Sometimes you may come across a tile service that uses the X, Y, zoom level tile schema to access tiles. If this is the case you can make use of the **GetTileUri** event handler on the **MapTileLayer**. When the **GetTileUri** event handler is fired the X, Y, and zoom level values are provided for the given tile and can be used generate the required tile URL. For example, WaymarkedTrails.org uses the following URL format to provide access to hiking routes that are generated by OpenStreetMap data as a tile layer.

http://tile.waymarkedtrails.org/hiking/{z}/{x}/{y}.png

To add this tile layer to the application open the **MainPage.xaml.cs** file add the following code to the **AddTileLayer_Tapped** button handler. This code will clear the map and add a new **MapTileLayer** that uses the **GetTileUri** event handler to get the URL to the hiking map using the x, y, and zoom level values.

C#

```
private void AddTileLayer_Tapped(object sender, Windows.UI.Xaml.Input.TappedRoutedEventArgs
e)
{
    ClearMap();

    MapTileLayer tileLayer = new MapTileLayer();
    tileLayer.GetTileUri += tileLayer_GetTileUri;
    MyMap.TileLayers.Add(tileLayer);
}

private void tileLayer_GetTileUri(object sender, GetTileUriEventArgs e)
{
    e.Uri = new Uri(string.Format("http://tile.waymarkedtrails.org/hiking/{0}/{1}/{2}.png",
e.LevelOfDetail, e.X, e.Y));
}
```

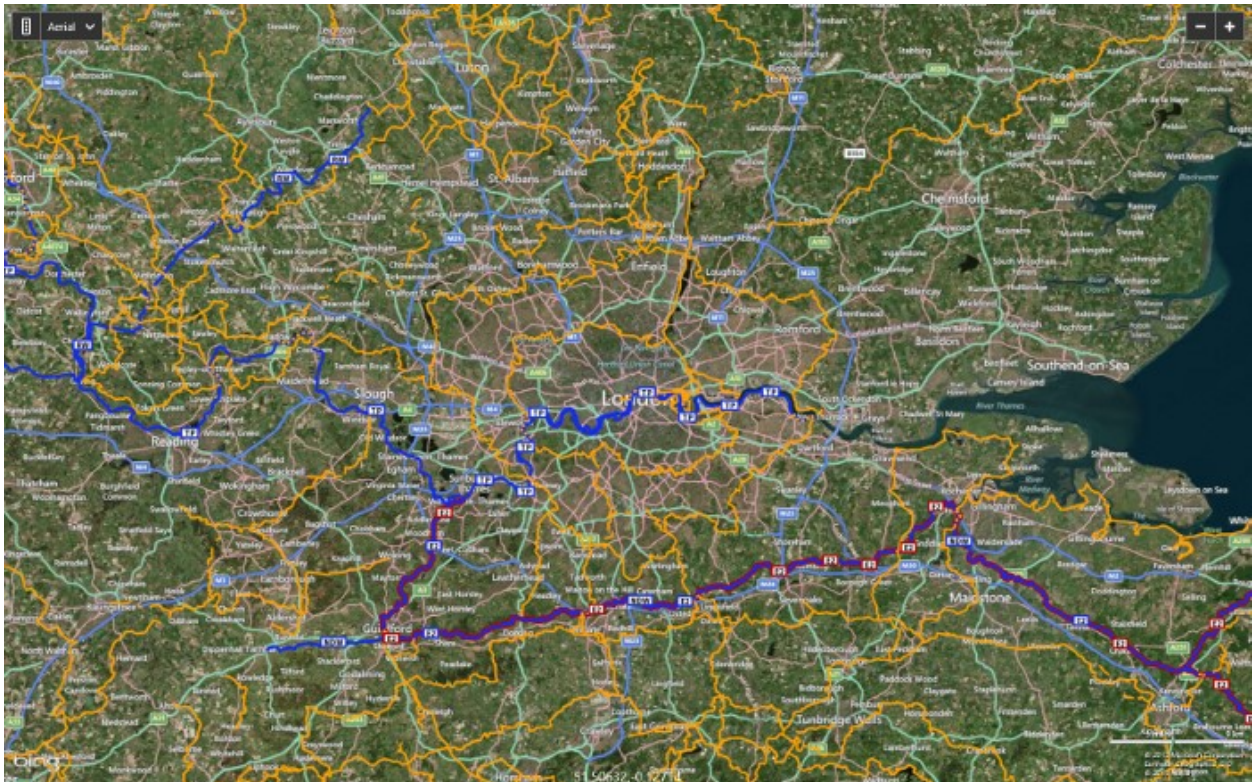
Visual Basic

```
Private Sub AddTileLayer_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
    ClearMap()

    Dim tileLayer = New MapTileLayer()
    AddHandler tileLayer.GetTileUri, AddressOf tileLayer_GetTileUri
    MyMap.TileLayers.Add(tileLayer)
End Sub

Private Sub tileLayer_GetTileUri(sender As Object, e As GetTileUriEventArgs)
    e.Uri = New Uri(String.Format("http://tile.waymarkedtrails.org/hiking/{0}/{1}/{2}.png",
e.LevelOfDetail, e.X, e.Y))
End Sub
```

If you click on this button you will see a bunch of colored lines that mark hiking trails appear on the map. Now if you pan and zoom the map you should find that the map moves smoothly, almost as if it didn't have any data at all on it.



There are many companies that expose mapping data in the form of a Web Mapping Service (WMS) or Web Map Tile Service (WMTS). These services can be used to generate tiles, however some of them may require additional information such as the bounding box of a tile. To ingest these types of services you can use the **GetTileUri** event handler of the **MapTileLayer**. When the **GetTileUri** event handler is fired you can calculate the bounding box of the tile and use this information to generate the required tile URL. The calculations are fairly complex. Here is an example of how to implement a WMS service that uses the bounding box of the tile in the URL:

C#

```
private void tileLayer_GetTileUri(object sender, GetTileUriEventArgs e)
{
    double north, south, east, west;
    TileXYZZoomToBBBox(e.X, e.Y, e.LevelOfDetail, out north, out south, out east, out west);

    string urlTemplate = "http://wms1.ccgis.de/cgi-bin/mapserv?map=/data/umn/germany/germany.map&&VERSION=1.1.1&REQUEST=GetMap&SERVICE=WMS&SRS=EPSG%3A4326&BBOX={0:N5},{1:N5},{2:N5},{3:N5}&WIDTH=256&HEIGHT=256&LAYERS=Topographie&format=image/png";

    e.Uri = new Uri(string.Format(urlTemplate, west, south, east, north));
}

private void TileXYZZoomToBBBox(int x, int y, int zoom, out double north, out double south, out double east, out double west)
{
    double mapSize = Math.Pow(2, zoom);

    west = ((x * 360) / mapSize) - 180;
    east = (((x + 1) * 360) / mapSize) - 180;

    double efactor = Math.Exp((0.5 - y / mapSize) * 4 * Math.PI);
```

```

north = (Math.Asin((efactor - 1) / (efactor + 1))) * (180 / Math.PI);

efactor = Math.Exp((0.5 - (y + 1) / mapSize) * 4 * Math.PI);
south = (Math.Asin((efactor - 1) / (efactor + 1))) * (180 / Math.PI);
}

```

Visual Basic

```

Private Sub tileLayer_GetTileUri(sender As Object, e As GetTileUriEventArgs)
    Dim north As Double, south As Double, east As Double, west As Double
    TileXYZZoomToBBox(e.X, e.Y, e.LevelOfDetail, north, south, east, west)

    Dim urlTemplate As String = "http://wms1.ccgis.de/cgi-
bin/mapserv?map=/data/umn/germany/germany.map&&VERSION=1.1.1&REQUEST=GetMap&SERVICE=WMS&SRS=
EPSG%3A4326&BBOX={0:N5},{1:N5},{2:N5},{3:N5}&WIDTH=256&HEIGHT=256&LAYERS=Topographie&format=
image/png"

    e.Uri = New Uri(String.Format(urlTemplate, west, south, east, north))
End Sub

Private Sub TileXYZZoomToBBox(x As Integer, y As Integer, zoom As Integer, ByRef north As
Double, ByRef south As Double, ByRef east As Double, ByRef west As Double)
    Dim mapSize As Double = Math.Pow(2, zoom)

    west = ((x * 360) / mapSize) - 180
    east = (((x + 1) * 360) / mapSize) - 180

    Dim efactor As Double = Math.Exp((0.5 - y / mapSize) * 4 * Math.PI)
    north = (Math.Asin((efactor - 1) / (efactor + 1))) * (180 / Math.PI)

    efactor = Math.Exp((0.5 - (y + 1) / mapSize) * 4 * Math.PI)
    south = (Math.Asin((efactor - 1) / (efactor + 1))) * (180 / Math.PI)
End Sub

```

Here are some great tile layer sources that you may find useful:

- OpenStreetMap derived services - <http://bit.ly/1aCamOA>
- Data.gov - <http://1.usa.gov/15jGq5C>
- USGS - <http://bit.ly/15RbIDA>
- Geoserver - <http://bit.ly/14LnTve>

Traffic Information

The Bing Maps SDK provides two types of traffic data. The first type is a tile layer that shows traffic flow data. This highlights roads with different colors to indicate what the flow of traffic is like compared to the speed limits on those roads. The traffic flow data can easily be turned on or off using the **ShowTraffic** property on the map.

The second type of traffic data is traffic incidents. Traffic incidents are point based data that represent things like road closures, accidents, and construction. Traffic incident data is available through the **TrafficManager** property of the map. The **TrafficManager** class has the following methods available:

Method	Description
ClearTrafficIncident	Clears all traffic incident pushpins from the map.
GetIncidentSymbol	Returns a description of the symbol used for the specified traffic incident type.
GetTrafficIncidentsAsync	Makes an asynchronous request for traffic incidents based on the specified request options.

ShowTrafficIncidents	Adds pushpins to the map representing the traffic incidents in the specified collection.
----------------------	--

To toggle the traffic data on and off on the map open the **MainPage.xaml.cs** file and add the following code to the **ToggleTraffic_Tapped** event handler. Notice that the **async** keyword is used by the event handler. The reason for this is that an asynchronous call will be made to get incident data from the web. The **await** keyword is also used with the asynchronous method call. This causes the code to wait for the method call to return a response before continuing to execute.

C#

```
private async void ToggleTraffic_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
    try
    {
        //Toggle the traffic layer on the map
        MyMap.ShowTraffic = !MyMap.ShowTraffic;

        if (MyMap.ShowTraffic)
        {
            // Request traffic incidents for the current view
            var requestOptions = new
Bing.Maps.Traffic.TrafficIncidentRequestOptions(MyMap.Bounds);

            var response = await
MyMap.TrafficManager.GetTrafficIncidentsAsync(requestOptions);

            // Display the traffic on the map
            MyMap.TrafficManager.ShowTrafficIncidents(response.TrafficIncidents);
        }
        else
        {
            //Remove traffic incidents
            MyMap.TrafficManager.ClearTrafficIncidents();
        }
    }
    catch { }
}
```

Visual Basic

```
Private Async Sub ToggleTraffic_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
    Try
        'Toggle the traffic layer on the map
        MyMap.ShowTraffic = Not MyMap.ShowTraffic

        If (MyMap.ShowTraffic) Then
            ' Request traffic incidents for the current view
            Dim requestOptions = New
Bing.Maps.Traffic.TrafficIncidentRequestOptions(MyMap.Bounds)

            Dim response = Await
MyMap.TrafficManager.GetTrafficIncidentsAsync(requestOptions)

            'Display the traffic on the map
            MyMap.TrafficManager.ShowTrafficIncidents(response.TrafficIncidents)
```

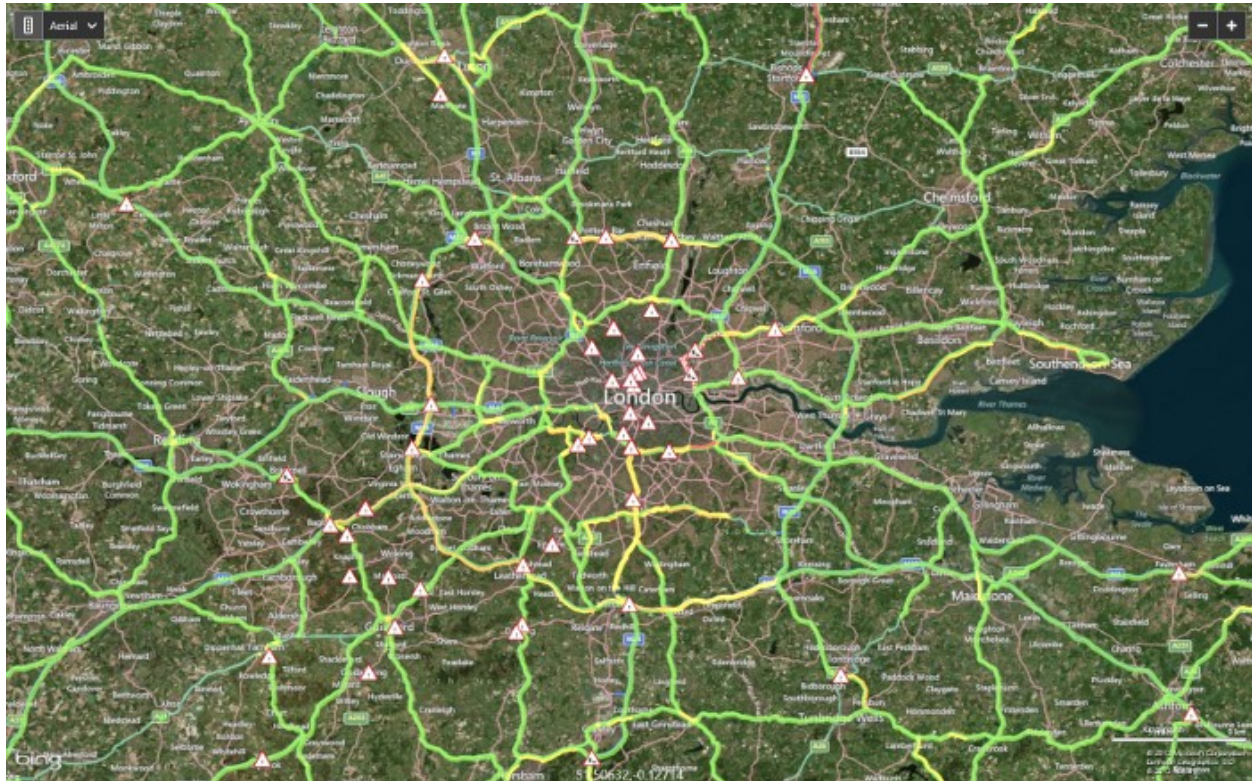


```

Else
    'Remove traffic incidents
    MyMap.TrafficManager.ClearTrafficIncidents()
End If
Catch
End Try
End Sub

```

If you click on this button traffic data will appear on the map.



Note that the incident data will only show up for the current map view and will not update as the map is moved. You could have this data update by using the **ShowTrafficIncidents** method when the **ViewChangeEnd** event fires on the map.

Traffic data is available in 34 countries. You can find a list of the currently supported regions here: <http://bit.ly/130ytgc>

Overlaying Custom User Controls

The native Bing Maps API provides you with the ability to add any **UIElement** directly to the map using the **MapLayer** class. This provides us with a lot more room for adding custom content to the map. Here are just a few simple examples of where you may want to add a custom user control to the map.

- Create a custom pushpin icon using an image.
- Add a custom label to the map using a **TextBlock**.
- Allow users to select countries using a check box.
- Create a Pie Chart and overlay on areas of the map to show relevant BI data.

Just like a pushpin you can add **UIElement**'s to the map or **MapLayer**'s using XAML or code. Here is an example of how to add a **TextBlock** to the map using XAML.

```
<m:MapLayer Name="DataLayer">
```

```

<TextBlock Text="Hello World">
    <m:MapLayer.Position>
        <m:Location Latitude="0" Longitude="0"/>
    </m:MapLayer.Position>
</TextBlock>
</m:MapLayer>

```

To do this through code open the **MainPage.xaml.cs** file and add the following code to the **AddUserControl_Tapped** button handler. This code will clear the map and add a **TextBlock** with the text "Hello World" to the center of the map.

C#

```

ClearMap();

//Create a simple User Control
TextBlock textBox = new TextBlock();
textBox.Text = "Hello World";
textBox.FontSize = 26;

MapLayer.SetPosition(textBox, MyMap.Center);
DataLayer.Children.Add(textBox);

```

Visual Basic

```

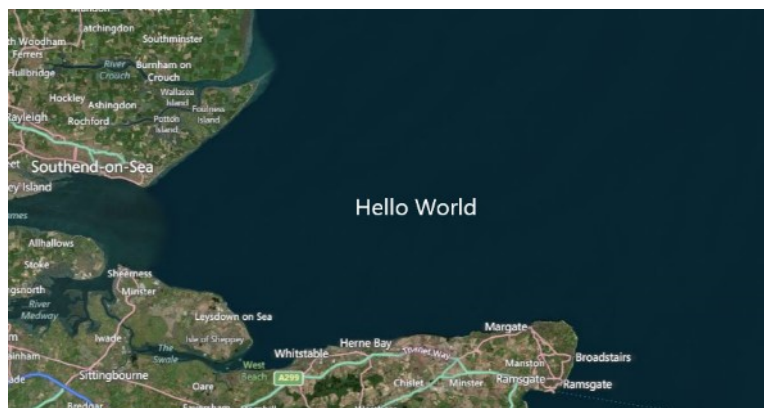
ClearMap()

'Create a simple User Control
Dim textBox As New TextBlock()
textBox.Text = "Hello World"
textBox.FontSize = 26

MapLayer.SetPosition(textBox, MyMap.Center)
DataLayer.Children.Add(textBox)

```

If you click on this button the text "Hello World" will display in the center of the map like so:



Note: In addition to adding custom user controls to the map you can easily wrap a map with a Grid control and position user controls over the map. This is useful for adding things like buttons and panels over the map display.

Creating Infoboxes

An infobox, also sometimes refer to as an info window, is a simple panel that displays information over top the map. This is often used to display information linked to a location after clicking on a pushpin. The Bing Maps native API does not come with an Infobox control. This might sound like a limitation or an oversight but in reality there was no need to create such a control. As we saw in the last section you can easily add any **UIElement** to the map. As such you can create a custom infobox and add it to the map easily.

When creating an infobox it is usually preferred to have it display above all the over data on the map. To achieve this it is a good practice to use a separate **MapLayer** which is added to the map after all other layers on the map as shown in the Layer Content section of this chapter. Another good practice is to only have one infobox control and use it for all your data. To do this update the **InfoboxLayer** XAML on the **MainPage.xaml** file like so.

```
<m:MapLayer Name="InfoboxLayer" Visibility="Collapsed">
    <Grid x:Name="Infobox">
        <Border Background="Black" Opacity="0.8" BorderBrush="White"
            BorderThickness="2" CornerRadius="5"/>

        <Grid Margin="5">
            <TextBlock Text="{Binding}" FontSize="20" Width="250"
                TextWrapping="Wrap" HorizontalAlignment="Left" />
            <Button Content="X" Tapped="CloseInfobox_Tapped"
                HorizontalAlignment="Right" VerticalAlignment="Top"/>
        </Grid>
    </Grid>
</m:MapLayer>
```

This XAML adds a hidden Grid control to the map which has a textbox for displaying information and a button for closing the infobox. The data in the textbox is populated by setting a string value as the data context of the infobox grid. In the **MainPage.xaml.cs** file add the following code which will add a pushpin to the map when you press the add infobox button. The pushpin will have a tapped event on it which will call the **OpenInfobox** method. In addition to this some metadata in the form of a string will be passed into the **Tag** property of the pushpin. The **OpenInfobox** method will position the Grid being used as an infobox such that it is near the pushpin. This method will also pass the metadata that is stored in the **Tag** property of the pushpin as the data context of the infobox. Finally, there is a **CloseInfobox_Tapped** event handler that will simply hide the infobox if fired.

C#

```
private void AddInfobox_Tapped(object sender, Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
    ClearMap();

    //Add a pushpin to the map
    Pushpin pin = new Pushpin();
    pin.Tag = "My Custom Infobox";
    MapLayer.SetPosition(pin, MyMap.Center);

    //Add a tap event to the pushpin to open the infobox
    pin.Tapped += OpenInfobox;

    //Add the pushpin to the DataLayer
    DataLayer.Children.Add(pin);
}

private void OpenInfobox(object sender, Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
    Pushpin pin = sender as Pushpin;
```

```

//Use the metadata from the pushpin to set the Data Context on the infobox
Infobox.DataContext = pin.Tag;

//Show the infobox layer
InfoboxLayer.Visibility = Visibility.Visible;

//Get the position of the pushpin and use it to position the infobox
MapLayer.SetPosition(Infobox, MapLayer.GetPosition(pin));
}

private void CloseInfobox_Tapped(object sender, Windows.UI.Xaml.Input.TappedRoutedEventArgs
e)
{
    InfoboxLayer.Visibility = Visibility.Collapsed;
}

```

Visual Basic

```

Private Sub AddInfobox_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
    ClearMap()

    ''Add a pushpin to the map
    Dim pin = New Pushpin()
    pin.Tag = "My Custom Infobox"
    MapLayer.SetPosition(pin, MyMap.Center)

    ''Add a tap event to the pushpin to open the infobox
    AddHandler pin.Tapped, AddressOf OpenInfobox

    ''Add the pushpin to the DataLayer
    DataLayer.Children.Add(pin)
End Sub

Private Sub OpenInfobox(sender As Object, e As Windows.UI.Xaml.Input.TappedRoutedEventArgs)
    Dim pin = TryCast(sender, Pushpin)

    ''Use the metadata from the pushpin to set the Data Context on the infobox
    Infobox.DataContext = pin.Tag

    ''Show the infobox layer
    InfoboxLayer.Visibility = Visibility.Visible

    ''Get the position of the pushpin and use it to position the infobox
    MapLayer.SetPosition(Infobox, MapLayer.GetPosition(pin))
End Sub

Private Sub CloseInfobox_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
    InfoboxLayer.Visibility = Visibility.Collapsed
End Sub

```

If you click on this button a pushpin will display on the map. If you click the pushpin an infobox will display with the text "My Custom Infobox" like so:



Search Manager

The map has a property called **SearchManager** which provides two key functionalities, geocoding and reverse geocoding. Geocoding is the process of taking an address and determining its location on the map. Reverse geocoding does the opposite and takes a coordinate and finds the closest known location. It does this by connecting to the Bing Maps REST services using an asynchronous method call. The **SearchManager** class has the following methods available:

Method	Description
GeocodeAsync	Makes an asynchronous geocode request based on the specified request options.
ReverseGeocodeAsync	Makes an asynchronous reverse geocode request based on the specified request options.

In order to geocode a location you need to pass in a **GeocodeRequestOptions** object into the **GeocodeAsync** method. The **GeocodeRequestOptions** class has the following properties that you can set.

Name	Type	Description
Bounds	LocationRect	A bounding box used to define an area to search for a location match.
IncludeNeighborhood	bool	A bool indicating whether to include neighborhood in the response, if it is available. The default value is false.
IncludeQueryParse	bool	A bool indicating whether to include information about how the QueryString was parsed. The default value is false.
MaxResults	int	The maximum number of results to return. The valid range is between 1 and 20. The default value is 5.
QueryString	String	The query string to geocode.

To implement the geocoding functionality open the **MainPage.xaml.cs** file and add the following code to the **GeocodeBtn_Tapped** event handler.

C#

```

private async void GeocodeBtn_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
    try
    {
        ClearMap();

        var options = new GeocodeRequestOptions("New York, NY");
        var response = await MyMap.SearchManager.GeocodeAsync(options);

        if (response.LocationData != null &&
            response.LocationData.Count > 0)
        {
            //Add pushpin to show geocoded location
            Pushpin pin = new Pushpin();
            MapLayer.SetPosition(pin, response.LocationData[0].Location);
            DataLayer.Children.Add(pin);

            //Position the map over the location
            MyMap.SetView(response.LocationData[0].Location, 15);
        }
        else
        {
            var dialog = new MessageDialog("No results found.", "Geocoding Result");
            await dialog.ShowAsync();
        }
    }
    catch { }
}

```

Visual Basic

```

Private Async Sub GeocodeBtn_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
    Try
        ClearMap()

        Dim options = New GeocodeRequestOptions("New York, NY")
        Dim response = Await MyMap.SearchManager.GeocodeAsync(options)

        If response.LocationData IsNot Nothing AndAlso response.LocationData.Count > 0 Then
            'Add pushpin to show geocoded location
            Dim pin As New Pushpin()
            MapLayer.SetPosition(pin, response.LocationData(0).Location)
            DataLayer.Children.Add(pin)

            'Position the map over the location
            MyMap.SetView(response.LocationData(0).Location, 15)
        Else
            Dim dialog = New MessageDialog("No results found.", "Geocoding Result")
            Await dialog.ShowAsync()
        End If
    Catch
    End Try
End Sub

```

If you press this button Bing Maps will geocode the locations "New York, NY", a pushpin will be added to the map for the first result and the map will zoom and center over the location.

To implement reverse geocoding add the following code to the **ReverseGeocodeBtn_Tapped** event handler in the **MainPage.xaml.cs** file.

C#

```
private async void ReverseGeocodeBtn_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
    try
    {
        ClearMap();

        //Reverse Geocode the center of the map.
        var options = new ReverseGeocodeRequestOptions(MyMap.Center);
        var response = await MyMap.SearchManager.ReverseGeocodeAsync(options);

        MessageDialog dialog;

        if (response.LocationData != null &&
            response.LocationData.Count > 0)
        {
            //Display the country the map is centered over
            dialog = new MessageDialog(response.LocationData[0].Address.CountryRegion,
"Reverse Geocoding Result");
        }
        else
        {
            dialog = new MessageDialog("Unable to reverse geocode location.", "Reverse
Geocoding Result");
        }

        await dialog.ShowAsync();
    }
    catch { }
}
```

Visual Basic

```
Private Async Sub ReverseGeocodeBtn_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
    Try
        ClearMap()

        'Reverse Geocode the center of the map.
        Dim options = New ReverseGeocodeRequestOptions(MyMap.Center)
        Dim response = Await MyMap.SearchManager.ReverseGeocodeAsync(options)

        Dim dialog As MessageDialog

        If response.LocationData IsNot Nothing AndAlso response.LocationData.Count > 0 Then
            'Display the country the map is centered over
            dialog = New MessageDialog(response.LocationData(0).Address.CountryRegion,
"Reverse Geocoding Result")
        Else
            dialog = New MessageDialog("Unable to reverse geocode location.", "Reverse
Geocoding Result")
        End If

        Await dialog.ShowAsync()
    End Try
End Sub
```

```

Catch
End Try
End Sub

```

If you press this button this method will reverse geocode the center of the map and if success it will display a message on the screen with the name of the country the map is centered over.

Directions Manager

The map has a property called **DirectionsManager** which the ability to calculate a route between multiple locations. This method makes use of the Bing Maps REST services to calculate the route by making an asynchronous connection. The **DirectionsManager** class has the following methods available:

Method	Description
CalculateDirectionsAsync	Makes an asynchronous request for route directions based on the set directions request and directions render options.
ClearActiveRoute	Clears the current route.
GetManeuverSymbol	Returns a description of the symbol used to represent the specified maneuver type.
GetMap	Returns the map on which to display the route.
HideRoutePath	Hides the specified route.
ShowDisambiguationPushpins	Displays disambiguation pushpins for the route stops on the map. If there is more than one disambiguated waypoint, show pushpins for the first waypoint.
ShowRoutePath	Displays the specified route on the map.

In order to route between locations you need to first add all the locations to the **Waypoints** property of the **DirectionsManager**. You can set options for how to calculate the route by setting the **RequestOptions** property of the map using a **DirectionsRequestOptions** object. Here are some of the more common properties of the **DirectionsRequestOptions** class.

Name	Type	Description
Avoid	AvoidOption	Features to avoid when calculating driving directions.
DistanceUnit	DistanceUnitOption	The distance unit to use when returning route distances.
FormattedInstruction	bool	A bool indicating whether to include formatted instructions in the route. Formatted instructions make it easier to display directions within a web page. The default value is false.
MaxSolutions	MaxSolutionsOption	The maximum number of transit or driving routes to return. This property is supported for the driving and transit routes between two waypoints in the United States, Canada, Mexico, United Kingdom, Australia and India.
Optimize	OptimizeOption	How the route calculation is optimized (i.e. By shortest distance). Note this causes the route to optimize the route while keeping the waypoints in their original order.
RouteMode	RouteModeOption	The mode of travel for the route; driving, walking, or transit.

Additionally you can set options for how the route is rendered on the map by setting the **RenderOptions** property of the map using a **DirectionsRenderOptions** object. Here are some of the more common properties of the **DirectionsRenderOptions** class.

Name	Type	Description
AutoDisplayDisambiguation	bool	A bool indicating whether to automatically display a disambiguation options for waypoint locations that

		need to be disambiguated. The default value is true. If this value is set to true, a DirectionsError event caused by waypoint disambiguation is not thrown.
AutoUpdateMapView	bool	A bool indicating whether to automatically set the map view to the best map view of the route. The default value is true.
DisambiguationPushpinOptions	DirectionsPushpinRenderOptions	The options used to render the pushpins that represent waypoints that need to be disambiguated.
DisplayItineraryItemHints	bool	A bool indicating whether to display route hints in the itinerary. Route hints provide additional information about following the route. The default value is true.
DisplayManeuverIcons	bool	A bool indicating whether to display icons for each direction maneuver. The default value is true.
DisplayStepWarnings	bool	A bool indicating whether to display available warnings for route directions. The default value is true.
DrivingPolylineOptions	DirectionsPolylineRenderOptions	The options used to render the polyline that represents a driving route on the map.
EndWaypointColorBrush	Brush	The color brush used to render the end waypoint of the route.
MiddleWaypointColorBrush	Brush	The color brush used to render stops along the route.
StartWaypointColorBrush	Brush	The color brush used to render the start waypoint of the route.
StepPushpinOptions	DirectionsPushpinRenderOptions	The options used to render the pushpins that represent directions along the route.

To calculate a simple route open the **MainPage.xaml.cs** file and add the following code to the **DirectionsBtn_Tapped** event handler. This will calculate the driving route between New York and Boston, optimized by shortest driving time and distances measured in miles.

C#

```
private async void DirectionsBtn_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
    try
    {
        ClearMap();

        //Specify waypoints to navigate route through
        var waypoints = new WaypointCollection();
        waypoints.Add(new Waypoint("New York, NY"));
        waypoints.Add(new Waypoint("Boston, MA"));

        MyMap.DirectionsManager.Waypoints = waypoints;

        //Set directions request options.
        var options = new DirectionsRequestOptions();
        options.Optimize = OptimizeOption.Time;
        options.RouteMode = RouteModeOption.Driving;
        options.DistanceUnit = DistanceUnitOption.Mile;

        MyMap.DirectionsManager.RequestOptions = options;
```

```

//Calculate directions
var response = await MyMap.DirectionsManager.CalculateDirectionsAsync();

if (response != null && response.Routes != null && response.Routes.Count > 0)
{
    MyMap.DirectionsManager.ShowRoutePath(response.Routes[0]);
}
else
{
    var dialog = new MessageDialog("Unable to calculate route.");
    await dialog.ShowAsync();
}
}
catch { }
}

```

Visual Basic

```

Private Async Sub DirectionsBtn_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
    Try
        ClearMap()

        'Specify waypoints to navigate route through
        Dim waypoints = New WaypointCollection()
        waypoints.Add(New Waypoint("New York, NY"))
        waypoints.Add(New Waypoint("Boston, MA"))

        MyMap.DirectionsManager.Waypoints = waypoints

        'Set directions request options.
        Dim options = New DirectionsRequestOptions()
        options.Optimize = OptimizeOption.Time
        options.RouteMode = RouteModeOption.Driving
        options.DistanceUnit = DistanceUnitOption.Mile

        MyMap.DirectionsManager.RequestOptions = options

        'Calculate directions
        Dim response = Await MyMap.DirectionsManager.CalculateDirectionsAsync()

        If response IsNot Nothing AndAlso response.Routes IsNot Nothing AndAlso
response.Routes.Count > 0 Then
            MyMap.DirectionsManager.ShowRoutePath(response.Routes(0))
        Else
            Dim dialog = New MessageDialog("Unable to calculate route.")
            Await dialog.ShowAsync()
        End If
    Catch
    End Try
End Sub

```

Next update the **ClearMap** method to hide the route line as shown below.

C#

```

private void ClearMap()
{
    DataLayer.Children.Clear();
}

```



```

PolyLayer.Shapes.Clear();
MyMap.TileLayers.Clear();

InfoboxLayer.Visibility = Visibility.Collapsed;

if (MyMap.DirectionsManager.ActiveRoute != null)
{
    MyMap.DirectionsManager.HideRoutePath(MyMap.DirectionsManager.ActiveRoute);
}
}

```

Visual Basic

```

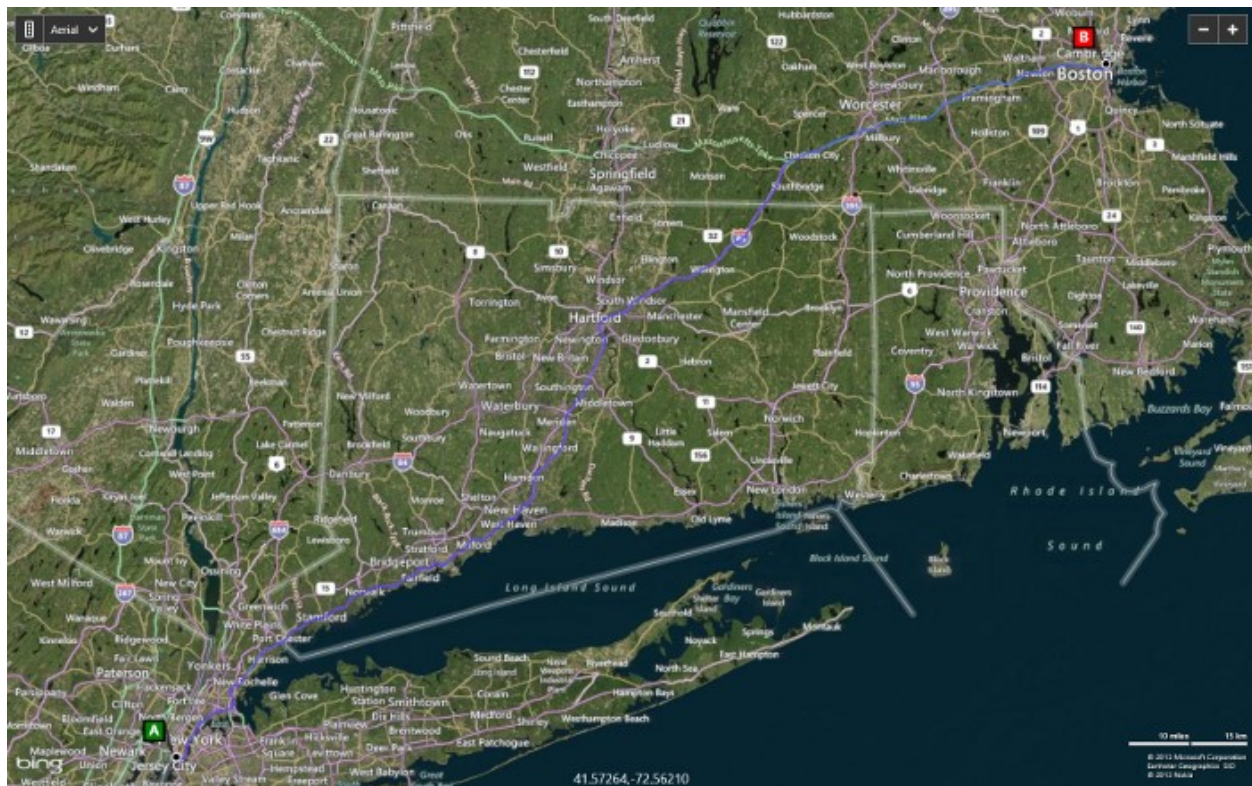
Private Sub ClearMap()
    DataLayer.Children.Clear()
    PolyLayer.Shapes.Clear()
    MyMap.TileLayers.Clear()

    InfoboxLayer.Visibility = Visibility.Collapsed

    If MyMap.DirectionsManager.ActiveRoute IsNot Nothing Then
        MyMap.DirectionsManager.HideRoutePath(MyMap.DirectionsManager.ActiveRoute)
    End If
End Sub

```

If you press the directions button the map will display a route from New York to Boston.



Venue Maps

Within Bing Maps you have the ability to view Venue Maps. Venue Maps are often thought of as maps of indoor structures such as malls and airports; however venue maps can be of just about any complex structure that is spread out over a large area. Some other types of venue maps available in Bing Maps include: the layout of shopping districts, stadiums, race courses, parks, and universities. Venue Maps are only displayed when the **MapType** is set to road imagery. Venues are accessed through the **VenueManager** property of the map. The **VenueManager** class has the following methods available for accessing venues.

Method	Description
ClearActiveVenue	Clears the active venue.
CreateVenueMapAsync	Creates a venue map.
GetNearbyVenuesAsync	Returns nearby venues.
GetAllCountriesAsync	Returns countries that have venue maps.
GetAllVenuesByCountryAsync	Returns all the venues in the specified country.
GetDirectoryPanel	Returns the directory panel.

In addition to these methods the **VenueManager** class also has the following properties.

Name	Type	Description
ActiveVenue	VenueMap	The venue that is active on the map.
ShowFloorControl	bool	A bool that indicates whether to show the floor control or not.
UseDefaultVenueOutline	bool	A bool that indicates whether a venue outline displays when the mouse moves over the venue or clicks on venue.
UseDefaultVenueTooltip	bool	A bool that indicates whether a tooltip displays when the mouse moves over the venue or clicks on venue.

There are a number of different ways to get a venue to display. One of the most common methods is to find nearby venues using the **GetNearbyVenuesAsync** method on the **VenueManager** property of the map. This method returns a **NearbyVenueCollection** which is a list of **NearbyVenue** objects. Nearby venues have a property called **Metadata** that contains all the metadata for the venue such as the name, location, and unique venue id. The venue id can be used passed to **CreateVenueMapAsync** method of the **VenueManager** class to retrieve the corresponding **Venue** object. This venue object can be set as the **ActiveVenue** on the **VenueManager** and its **Bounds** property used to zoom into the best view of the venue. The following code can be used in the **LoadVenueMap_Tapped** event to search within a 1,000 meters of the center of the map for a venue and load the first one that is found.

C#

```
private async void LoadVenueMap_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
    try
    {
        ClearMap();

        //Set the map type to road as Venue Maps are only displayed in road view.
        MyMap.MapType = MapType.Road;

        //Search for venues that are within 1000 meters (1km) of the center of the map.
        var nearbyVenueOptions = new Bing.Maps.VenueMaps.NearbyVenueOptions(MyMap.Center,
1000);

        var response = await MyMap.VenueManager.GetNearbyVenuesAsync(nearbyVenueOptions);
```



```

        if (response != null && response.Count > 0)
        {
            //Find the closest nearby venue, and set it as the ActiveVenue
            MyMap.VenueManager.ActiveVenue = await
MyMap.VenueManager.CreateVenueMapAsync(response[0].Metadata.VenueId);

            if (MyMap.VenueManager.ActiveVenue != null)
            {
                //Pan/zoom the map to the best map view for this venue.
                MyMap.SetView(MyMap.VenueManager.ActiveVenue.Bounds);
            }
        }
        else
        {
            var dialog = new MessageDialog("No Venue Maps found.");
            await dialog.ShowAsync();
        }
    }
    catch { }
}

```

Visual Basic

```

Private Async Sub LoadVenueMap_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
    Try
        ClearMap()

        ''Set the map type to road as Venue Maps are only displayed in road view.
        MyMap.MapType = MapType.Road

        ''Search for venues that are within 1000 meters (1km) of the center of the map.
        Dim nearbyVenueOptions = New Bing.Maps.VenueMaps.NearbyVenueOptions(MyMap.Center,
1000)

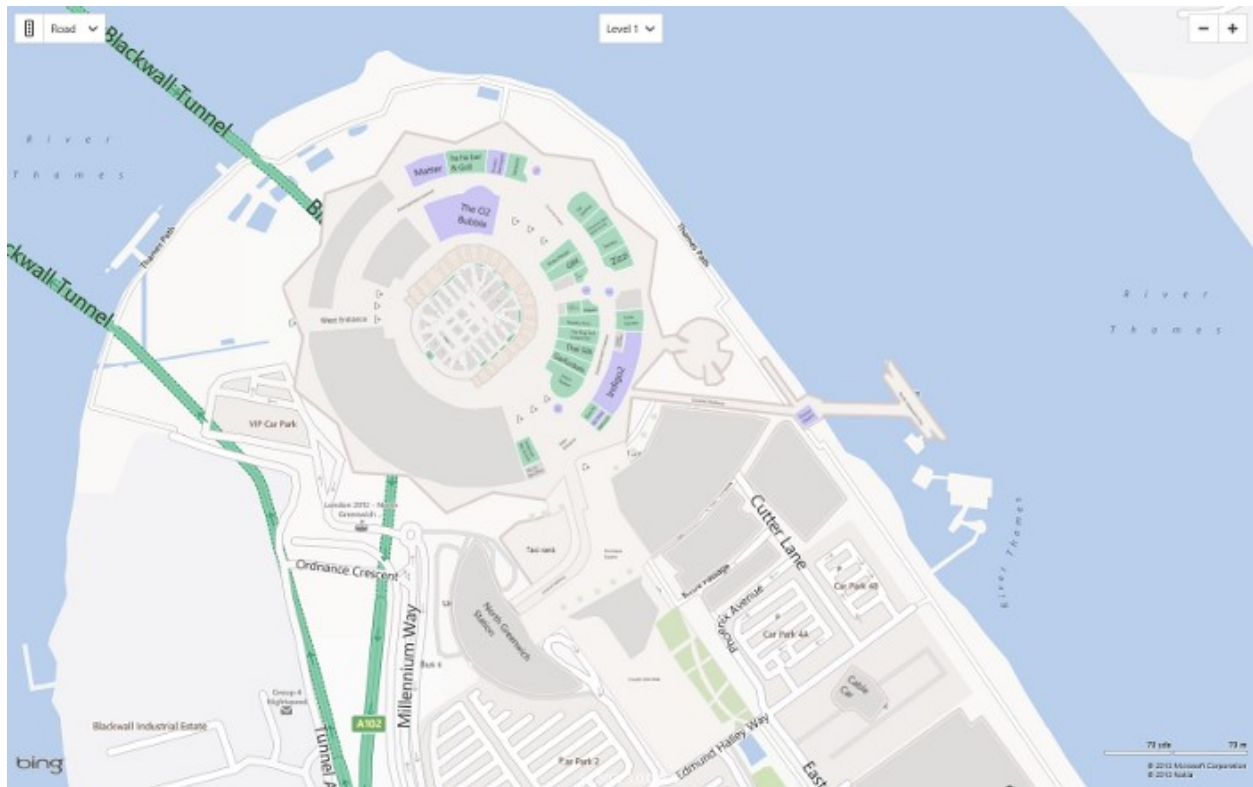
        Dim response = Await MyMap.VenueManager.GetNearbyVenuesAsync(nearbyVenueOptions)

        If response IsNot Nothing AndAlso response.Count > 0 Then
            ''Find the closest nearby venue, and set it as the ActiveVenue
            MyMap.VenueManager.ActiveVenue = Await
MyMap.VenueManager.CreateVenueMapAsync(response(0).Metadata.VenueId)

            If MyMap.VenueManager.ActiveVenue IsNot Nothing Then
                ''Pan/zoom the map to the best map view for this venue.
                MyMap.SetView(MyMap.VenueManager.ActiveVenue.Bounds)
            End If
        Else
            Dim dialog = New MessageDialog("No Venue Maps found.")
            Await dialog.ShowAsync()
        End If
    Catch
    End Try
End Sub

```

If the venue map button is pressed and a venue exists near the center of the map, the map will zoom into the venue and the inside of the venue map will be displayed.



Working with Map Events

The Bing Maps control provides many events to allow your application to respond to user actions. The **EntityCollection**, **Map**, **Pushpin**, **Polyline**, and **Polygon** classes along with many of the modules all have events. Here are a couple of examples of where you may use events:

- If a user clicks on a shape, trigger an event that opens an infobox.
- If the map style changes you may want to change the color of items overlaid on the map so that they stand out better. Light colors on the aerial maps and dark colors on the road maps.
- If the user has moved the map you may want to load in new data for the current map view once the user has finished moving the map.

There are a number of events available to the map, here are the most commonly used events:

Name	Description
CopyrightChanged	Occurs when the copyright of the map changes.
LandmarkTapped	Occurs when a landmark is tapped.
MapStyleChanged	Occurs when the map style changes.
TargetViewChanged	Occurs when the view towards which the map is navigating changes.
TileDownloadCompleted	Occurs when all of the map tiles of the map view have downloaded.
TileServersAvailabilityChanged	Occurs when the connection to the tile servers is lost or restored.
ViewChanged	Occurs when the map view changes. This event is raised for every frame of the map view change.
ViewChangeEnded	Occurs when the map view is done changing.
ViewChangeStarted	Occurs when the map view starts changing.

The Map class uses the inherited **UIElement** events. Some **UIElement** events have been overridden by the Map class. For these events, which are listed below, use the override event.

Name	Description
DoubleTappedOverride	Occurs when an otherwise unhandled DoubleTap interaction occurs over the map.
KeyDownOverride	Occurs when a keyboard key is pressed while the map has focus.
KeyUpOverride	Occurs when a keyboard key is released while the map has focus.
PointerCanceledOverride	Occurs when a pointer that made contact abnormally loses contact.
PointerCaptureLostOverride	Occurs when pointer capture previously held by the map moves to another element or elsewhere.
PointerEnteredOverride	Occurs when a pointer enters the hit test area of the map.
PointerExitedOverride	Occurs when a pointer leaves the hit test area of the map.
PointerMovedOverride	Occurs when a pointer moves while the pointer remains within the hit test area of the map.
PointerPressedOverride	Occurs when the pointer device initiates a Press action within the map.
PointerReleasedOverride	Occurs when the pointer device that previously initiated a Press action is released, while within the map.
PointerWheelChangedOverride	Occurs when the delta value of a pointer wheel changes.
TappedOverride	Occurs when an otherwise unhandled Tap interaction occurs over the hit test area of the map.

To test out the **ViewChange** event open up the **MainPage.xaml** file and add the following XAML to the **Grid** control after the map.

```
<Border Background="Black" HorizontalAlignment="Center" VerticalAlignment="Bottom">
    <TextBlock Name="CoordinatesTbx" FontSize="24" Margin="10"/>
</Border>
```

Open the **MainPage.xaml.cs** file and attach an event handler to the map's **ViewChanged** event to the **MainPage** constructor. Add the **MyMap_ViewChanged** event handler to the file.

C#

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

    PolyLayer = new MapShapeLayer();
    MyMap.ShapeLayers.Add(PolyLayer);

    MyMap.ViewChanged += MyMap_ViewChanged;
}

private void MyMap_ViewChanged(object sender, ViewChangedEventArgs e)
{
    CoordinatesTbx.Text = string.Format("{0:N5},{1:N5}", MyMap.Center.Latitude,
    MyMap.Center.Longitude);
}
```

Visual Basic

```
Public Sub New()
    InitializeComponent()
```

```

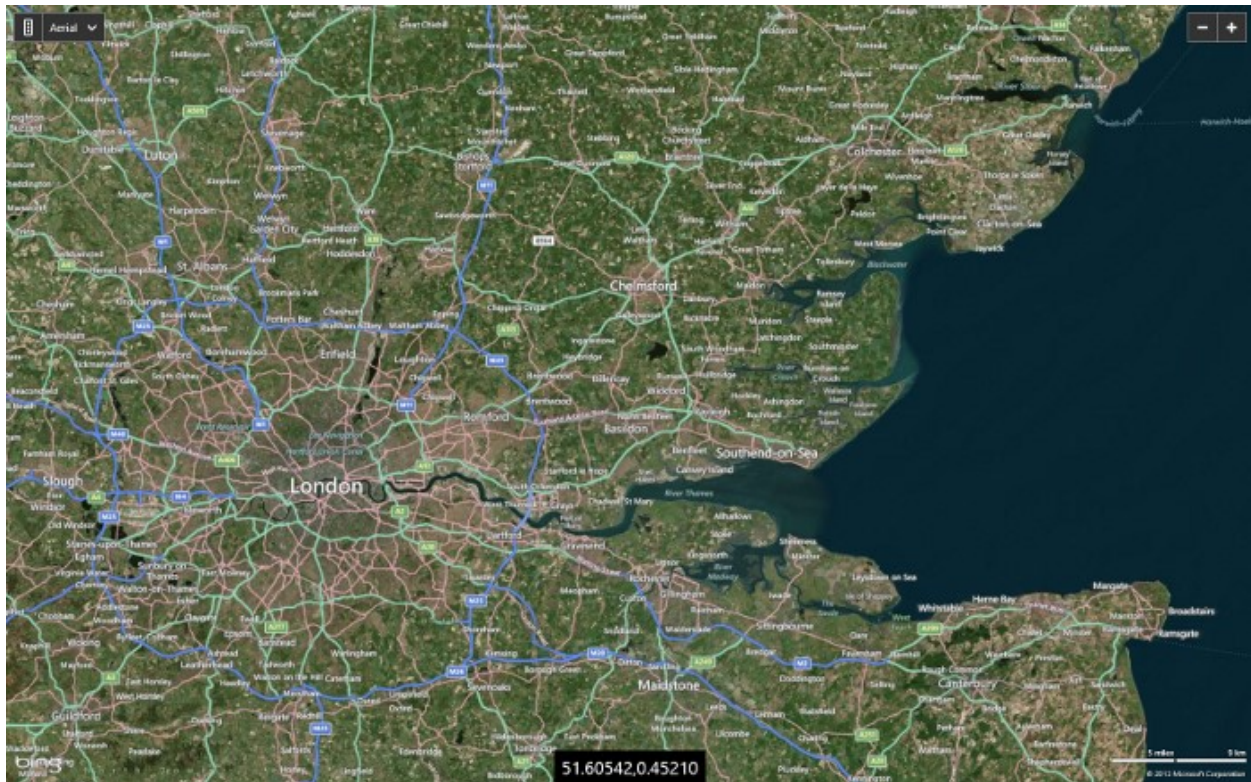
PolyLayer = New MapShapeLayer()
MyMap.ShapeLayers.Add(PolyLayer)

AddHandler MyMap.ViewChanged, AddressOf MyMap_ViewChanged
End Sub

Private Sub MyMap_ViewChanged(sender As Object, e As ViewChangedEventArgs)
    CoordinatesTbx.Text = String.Format("{0:N5},{1:N5}", MyMap.Center.Latitude,
MyMap.Center.Longitude)
End Sub

```

As you pan the map you will notice that the coordinate information for the center of the map will appear at the bottom of the screen.

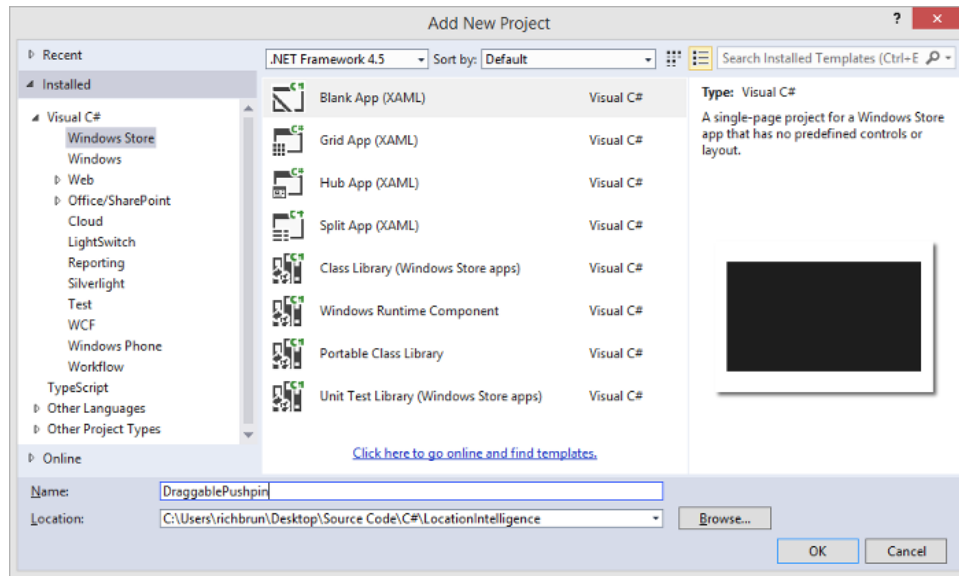


Creating Draggable Pushpins

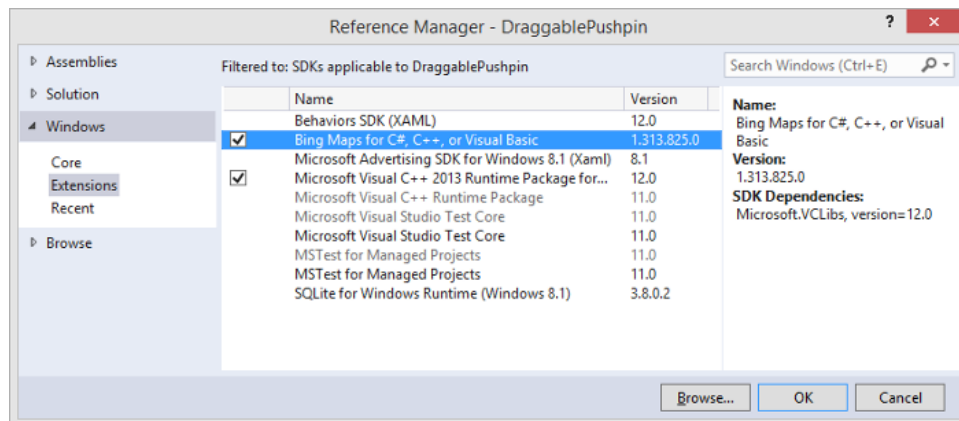
Depending on the type of application you are building you may want to give the user the ability to drag a pushpin. In the JavaScript version of Bing Maps this can easily be done simply by setting the **draggable** property of a pushpin to true. The **Pushpin** class in the native version of Bing Maps does not have this property. This isn't really a bad thing though as we can easily create a reusable user control that gives us a lot more flexibility in terms of customization. Some common uses for draggable pushpins include:

- Allowing the user to edit locations on the map simply by updating the position of a pushpin.
- Use them as handles for data points in drawing tools.

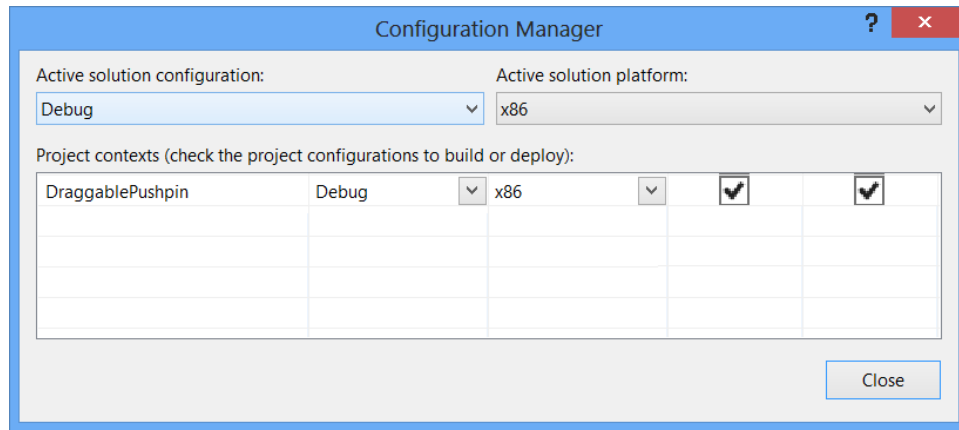
To get started open up Visual Studio and create a new project in your preferred language: C# or Visual Basic. Select the **Blank App** template and call the application **DraggablePushpin** and press OK.



Add a reference to the Bing Maps SDK. To do this right click on the **References** folder and press **Add Reference**. Select **Windows -> Extensions**, and then select **Bing Maps for C#, C++ and Visual Basic**. If you do not see this option, be sure to verify that you have installed the Bing Maps SDK for Windows Store apps. While you are here, also add a reference to the **Microsoft Visual C++ Runtime Package** as this is required by the Bing Maps SDK when developing using C# or Visual Basic.



Set the **Active solution platform** in Visual Studio by right clicking on the Solution folder and select **Properties**. Then go to **Configuration Properties -> Configuration**. Find your project and under the **Platform** column set the target platform to x86 and press OK.



Right click on the solution and select **Add -> New Item**. Select the **User Control** template and call it **DraggablePin**. Open the **DraggablePin.xaml** file and update the XAML to the following:

```
<UserControl
    x:Class="DraggablePushpin.DraggablePin"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:DraggablePushpin"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006">

    <Ellipse Height="30" Width="30" Fill="Blue"
        Stroke="White" StrokeThickness="5" Margin="-15,-15,0,0"/>
</UserControl>
```

The code behind for this user control will need to take in a reference to the map control in its constructor and a local reference to it stored in a private variable. This control will use a number of events on the map to enable the dragging functionality. We will have to override the **OnPointerPressed** event handler and use this to add the required map events for dragging the user control. When the user control is initially pressed the coordinate of the center of the map will be stored. The **ViewChange** event of the map will be used to set the center of the map to the stored center value. This will disable the panning of the map which we don't want to occur when dragging the user control. The **PointerMovedOverride** event of the map will be used to update the position of the user control as it is dragged. The **OnPointerReleasedOverride** event of the map will be used to release all the events from the map that were used for dragging the user control. In addition to this we will add a **Boolean** property called **Draggable** which can be used to disable the dragging functionality of the pushpin. We will also expose three **Action** events; **DragStart**, **Drag**, and **DragEnd**. These events will prove useful in future applications. The following code does all of this, add this to the **DraggablePin.xaml.cs** file.

C#

```
using Bing.Maps;
using System;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Input;

namespace DraggablePushpin
{
    public sealed partial class DraggablePin : UserControl
    {
        private Map _map;
        private bool isDragging = false;
```

```

Location _center;

public DraggablePin(Map map)
{
    this.InitializeComponent();

    _map = map;
}

/// <summary>
/// A boolean indicating whether the pushpin can be dragged.
/// </summary>
public bool Draggable { get; set; }

/// <summary>
/// Occurs when the pushpin is being dragged.
/// </summary>
public Action<Location> Drag;

/// <summary>
/// Occurs when the pushpin starts being dragged.
/// </summary>
public Action<Location> DragStart;

/// <summary>
/// Occurs when the pushpin stops being dragged.
/// </summary>
public Action<Location> DragEnd;

protected override void OnPointerPressed(PointerRoutedEventArgs e)
{
    base.OnPointerPressed(e);

    if (Draggable)
    {
        if (_map != null)
        {
            //Store the center of the map
            _center = _map.Center;

            //Attach events to the map to track touch and movement events
            _map.ViewChanged += Map_ViewChanged;
            _map.PointerReleasedOverride += Map_PointerReleased;
            _map.PointerMovedOverride += Map_PointerMoved;
        }

        var pointerPosition = e.GetCurrentPoint(_map);

        Location location = null;

        //Convert the point pixel to a Location coordinate
        if (_map.TryPixelToLocation(pointerPosition.Position, out location))
        {
            MapLayer.SetPosition(this, location);
        }

        if (DragStart != null)
        {
            DragStart(location);
        }
    }
}

```

```

        //Enable Dragging
        this.isDragging = true;
    }
}

private void Map_PointerMoved(object sender, PointerRoutedEventArgs e)
{
    //Check if the user is currently dragging the Pushpin
    if (this.isDragging)
    {
        //If so, move the Pushpin to where the pointer is.
        var pointerPosition = e.GetCurrentPoint(_map);

        Location location = null;

        //Convert the point pixel to a Location coordinate
        if (_map.TryPixelToLocation(pointerPosition.Position, out location))
        {
            MapLayer.SetPosition(this, location);
        }

        if (Drag != null)
        {
            Drag(location);
        }
    }
}

private void Map_PointerReleased(object sender, PointerRoutedEventArgs e)
{
    //Pushpin released, remove dragging events
    if (_map != null)
    {
        _map.ViewChanged -= Map_ViewChanged;
        _map.PointerReleasedOverride -= Map_PointerReleased;
        _map.PointerMovedOverride -= Map_PointerMoved;
    }

    var pointerPosition = e.GetCurrentPoint(_map);

    Location location = null;

    //Convert the point pixel to a Location coordinate
    if (_map.TryPixelToLocation(pointerPosition.Position, out location))
    {
        MapLayer.SetPosition(this, location);
    }

    if (DragEnd != null)
    {
        DragEnd(location);
    }

    this.isDragging = false;
}

private void Map_ViewChanged(object sender, ViewChangedEventArgs e)
{
    if (isDragging)
    {
        //Reset the map center to the stored center value.
    }
}

```



```

        //This prevents the map from panning when we drag across it.
        _map.Center = _center;
    }
}
}
}

```

Visual Basic

Imports Bing.Maps

Public NotInheritable Class DraggablePin
Inherits UserControl

Private _map As Map
Private isDragging As Boolean = False
Private _center As Location

Public Sub New(map As Map)
Me.InitializeComponent()

_map = map
End Sub

'''<summary>
''' A boolean indicating whether the pushpin can be dragged.
'''</summary>

Public Property Draggable() As Boolean
Get
Return m_Draggable
End Get
Set(value As Boolean)
m_Draggable = Value
End Set
End Property
Private m_Draggable As Boolean

''' <summary>
''' Occurs when the pushpin is being dragged.
''' </summary>
Public Drag As Action(Of Location)

''' <summary>
''' Occurs when the pushpin starts being dragged.
''' </summary>
Public DragStart As Action(Of Location)

''' <summary>
''' Occurs when the pushpin stops being dragged.
''' </summary>
Public DragEnd As Action(Of Location)

Protected Overrides Sub OnPointerPressed(e As PointerRoutedEventArgs)
MyBase.OnPointerPressed(e)

If Draggable Then
If _map IsNot Nothing Then
''Store the center of the map
_center = _map.Center

```

        ''Attach events to the map to track touch and movement events
        AddHandler _map.ViewChanged, AddressOf Map_ViewChanged
        AddHandler _map.PointerReleasedOverride, AddressOf Map_PointerReleased
        AddHandler _map.PointerMovedOverride, AddressOf Map_PointerMoved
    End If

    Dim pointerPosition = e.GetCurrentPoint(_map)

    Dim location As Location = Nothing

    ''Convert the point pixel to a Location coordinate
    If _map.TryPixelToLocation(pointerPosition.Position, location) Then
        MapLayer.SetPosition(Me, location)
    End If

    If DragStart IsNot Nothing Then
        DragStart(location)
    End If

    ''Enable Dragging
    Me.isDragging = True
End If
End Sub

Private Sub Map_PointerMoved(sender As Object, e As PointerRoutedEventArgs)
    ''Check if the user is currently dragging the Pushpin
    If Me.isDragging Then
        ''If so, move the Pushpin to where the pointer is.
        Dim pointerPosition = e.GetCurrentPoint(_map)

        Dim location As Location = Nothing

        ''Convert the point pixel to a Location coordinate
        If _map.TryPixelToLocation(pointerPosition.Position, location) Then
            MapLayer.SetPosition(Me, location)
        End If

        If Drag IsNot Nothing Then
            Drag(location)
        End If
    End If
End Sub

Private Sub Map_PointerReleased(sender As Object, e As PointerRoutedEventArgs)
    ''Pushpin released, remove dragging events
    If _map IsNot Nothing Then
        RemoveHandler _map.ViewChanged, AddressOf Map_ViewChanged
        RemoveHandler _map.PointerReleasedOverride, AddressOf Map_PointerReleased
        RemoveHandler _map.PointerMovedOverride, AddressOf Map_PointerMoved
    End If

    Dim pointerPosition = e.GetCurrentPoint(_map)

    Dim location As Location = Nothing

    ''Convert the point pixel to a Location coordinate
    If _map.TryPixelToLocation(pointerPosition.Position, location) Then
        MapLayer.SetPosition(Me, location)
    End If

    If DragEnd IsNot Nothing Then

```

```

        DragEnd(location)
    End If

    Me.isDragging = False
End Sub

Private Sub Map_ViewChanged(sender As Object, e As ViewChangedEventArgs)
    If isDragging Then
        'Reset the map center to the stored center value.
        'This prevents the map from panning when we drag across it.
        _map.Center = _center
    End If
End Sub
End Class

```

Now that we have a **DraggablePin** user control we can now implement it. Open the **MainPage.xaml.cs** file and update the XAML to the following.

```

<Page
    x:Class="DraggablePushpin.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:DraggablePushpin"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:m="using:Bing.Maps">

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <m:Map Name="MyMap" Credentials="YOUR_BING_MAPS_KEY"/>

        <Border Background="Black" HorizontalAlignment="Center" VerticalAlignment="Bottom">
            <TextBlock Name="CoordinatesTbx" FontSize="24" Margin="10"/>
        </Border>
    </Grid>
</Page>

```

In the **MainPage.xaml.cs** file we will attach to the **MapLoaded** event of the map. When this event fires we will create a **DraggablePin** and add it to the center of the map. We will also add attach to the **Drag** event of the pin and have it update the coordinates displayed in a **TextBlock** at the bottom of the map. The following code can be used in the **MainPage.xaml.cs** file to accomplish this.

C#

```

using Windows.UI.Xaml.Controls;

namespace DraggablePushpin
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();

            MyMap.Loaded += MyMap_Loaded;
        }

        private void MyMap_Loaded(object sender, Windows.UI.Xaml.RoutedEventArgs e)
        {
            DraggablePin pin = new DraggablePin(MyMap);

```

```

        //Set the location of the pin to the center of the map.
        Bing.Maps.MapLayer.SetPosition(pin, MyMap.Center);

        //Set the pin as draggable.
        pin.Draggable = true;

        //Attach to the drag action of the pin.
        pin.Drag += Pin_Dragged;

        //Add the pin to the map.
        MyMap.Children.Add(pin);
    }

    private void Pin_Dragged(Bing.Maps.Location location)
    {
        CoordinatesTbx.Text = string.Format("{0:N5},{1:N5}", location.Latitude,
location.Longitude);
    }
}

```

Visual Basic

```

Imports Windows.UI.Xaml.Controls

Public NotInheritable Class MainPage
    Inherits Page

    Public Sub New()
        InitializeComponent()

        AddHandler MyMap.Loaded, AddressOf MyMap_Loaded
    End Sub

    Private Sub MyMap_Loaded(sender As Object, e As Windows.UI.Xaml.RoutedEventArgs)
        Dim pin As New DraggablePin(MyMap)

        ''Set the location of the pin to the center of the map.
        Bing.Maps.MapLayer.SetPosition(pin, MyMap.Center)

        ''Set the pin as draggable.
        pin.Draggable = True

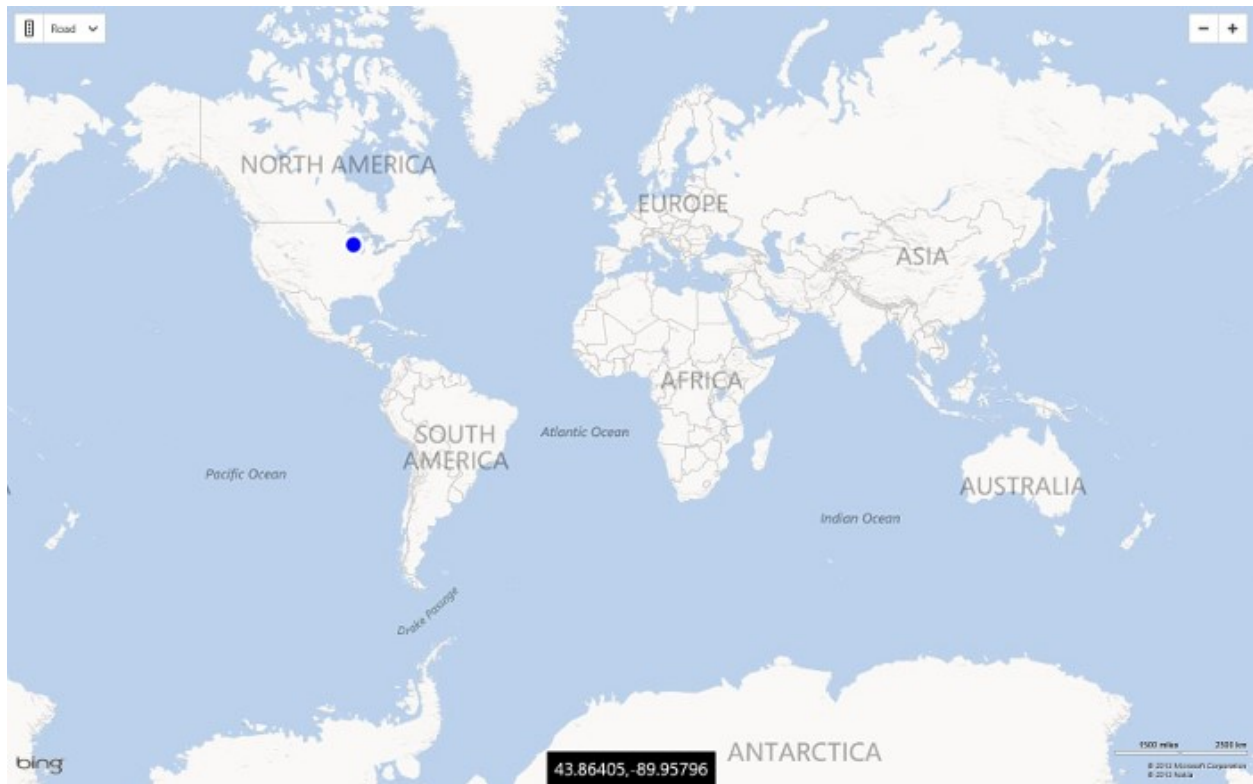
        ''Attach to the drag action of the pin.
        pin.Drag = AddressOf Pin_Dragged

        ''Add the pin to the map.
        MyMap.Children.Add(pin)
    End Sub

    Private Sub Pin_Dragged(location As Bing.Maps.Location)
        CoordinatesTbx.Text = String.Format("{0:N5},{1:N5}", location.Latitude,
location.Longitude)
    End Sub
End Class

```

Now if you run the application you will see a blue circle pushpin in the center of the map. Simply press down on it and drag it around on the screen using a mouse or your finger if you have a touch screen device.



Chapter Summary

In this chapter you were introduced to native Bing Maps Windows Store. Here is a short summary of some key points from this chapter.

- When adding a reference to the Bing Maps SDK in a C# or Visual Basic project you have to also add a reference to the Microsoft Visual C++ Runtime Package. As such you must also set the **Active solution platform** in the Configuration Manager as x86, x64 or ARM.
- Bing Maps supports 116 languages. A full list of supported cultures can be found here: <http://bit.ly/1aY3gDh>.
- **Pushpins** and Controls that inherit from the **UIElement** class can be added to **MapLayer**'s.
- The position of a **Pushpin** or a **UIElement** can be set by using the **MapLayer.SetPosition** method.
- **MapLayer**'s are added to the **Children** property of the map.
- **MapPolygon**'s and **MapPolyline**'s can be added to the map using **MapShapeLayer**'s.
- **MapShapeLayer**'s should be added to the map through code if you wish to add shapes to the layer programmatically.
- The Pushpin class inherits from the **UIElement** class, as such it has a property called **Tag** which is an **object** type. This property is handy for storing metadata that is related to the pushpin such as a unique identifier, or the data needed to populate an infobox.
- It is a best practice to separate your data and infoboxes by using two layers. By doing this you will be able to ensure that your infobox is always rendered above the pushpins on the map.
- Tile layers are an excellent way to visualize large data sets on a map. Often the size of an image of the data is much smaller than the data itself. The smaller size means faster downloads and less objects for the map to render which makes for a better performance. Some good sources for tile layers include:
 - OpenStreetMap derived services - <http://bit.ly/1aCamOA>
 - Data.gov - <http://1.usa.gov/15jGq5C>
 - USGS - <http://bit.ly/15RbIDA>

- Geoserver - <http://bit.ly/14LnTve>
- Methods that contain code which make asynchronous method calls will need to use the ***async*** keyword in the method definition and the ***await*** keyword with the asynchronous method call. If the ***await*** keyword is not used the code will continue to execute before the asynchronous method call has returned a response.
- The map has a property called ***SearchManager*** which provides geocoding and reverse geocoding functionalities. Geocoding takes an address and determines its location on the map. Reverse geocoding finds the closest known location to a coordinate.
- The map has a property called ***DirectionsManager*** which provides the tools needed for calculating routes between locations for driving, walking and public transport.
- Traffic data is available through the ***TrafficManager*** property of the map. Traffic flow and traffic incident data are both available. A list of countries where traffic data is available can be found here: <http://bit.ly/130ytc>
- The map has a property called ***VenueManager*** which provides the tools needed to find and load Venue Maps. Venue Maps are often used to show indoor floor plans of complex buildings such as malls or airports.

Chapter 5: Bing Maps REST Services

There are several services available through Bing Maps that provide a number of powerful features. The two main services are the Bing Maps REST Services and the Bing Spatial Data Services. Both of these services are REST based services. The acronym REST stands for Representational State Transfer. A REST service is a URL based web service that makes use of the built-in standards within HTTP for such things as authentication and content type negotiation. The most common methods of communicating with a REST service is through HTTP GET and POST requests. These types of services tend to be much quicker, smaller and easier to use across a number of different programming languages when compared to traditional SOAP based services.

Both of these REST services are capable of returning responses in JSON or XML format. The XML responses from REST tend to be smaller than equivalent SOAP XML responses because SOAP services “wrap” responses with additional service-related information. The smaller size of REST responses generally means faster response times between the client and server. In this chapter we will only work with the JSON responses as they are much smaller and faster to download than the XML response.

The Bing Maps services and content is hosted in Microsoft’s CDN and dynamic compute centers. This enables end users to access these services and content from data centers closest to their location for lower latency and faster responses while simultaneously increasing the overall stability and scalability of all services.

You can use these services to access a lot of the Bing Maps content directly. This can be useful in applications where you may just need the driving distance between two locations, or the address of a GPS location, or to generate an image of a map if does not need to be interactive. In these types of situations you can use the services directly without needing to load the full interactive Bing Maps control and thus significantly reduce the resources used by your application.

Most of these services are accessed using standard HTTP Get requests. One nice benefit of this is that you can easily test these services by simply loading a URL in a standard browser. One drawback to using HTTP GET requests is that browsers have a limit on how long a URL can be. In general URL’s should not exceed 2,083 characters in length. There are a couple of features in these API’s that allow you to send over large amounts of data. To get around the URL length limitations HTTP POST requests are used.

Important Note: You may come across information about the Bing Maps SOAP services online and may consider using it. This is an old legacy service that is significantly slower and less accurate than the Bing Maps REST services.

Bing Maps REST Services Overview

This Bing Maps REST Services exposes a number of API’s for performing queries against the Bing Maps content. Here is a list of the API’s available in this service.

API	Description
Locations	Provides geocoding and reverse-geocoding functionality.
Elevations	Provides elevation data information for all point around the globe.
Imagery	Generates static map images and provides imagery metadata information.
Routes	Generates directions and route information for driving, walking or using transit.
Traffic	Provides information about traffic incidents and issues in a specified area.

In this book we will focus on the Locations, Imagery and Routes API's. These API's are accessed by using a common URL format:

`http://dev.virtualearth.net/REST/v1/[API]?[SearchParameters]&key=BingMapsKey`

The two main parts of this common URL that you will need to include is the name of the API and the search parameters for the selected API. Note that all requests to the Bing Maps REST services require a Bing Maps key. In addition to the search parameters that are specific to each API there are also a common set of search parameters available to all API's:

Parameter	Description
key	A Bing Maps key used to authenticate the service request. (Required)
culture	The culture to use for the request. A full list of supported cultures can be found here http://bit.ly/14rWkIM
output	The output format of the response. (Optional) <ul style="list-style-type: none"> • json (default) • xml
jsonp	Name of a JSON callback function that is called when the response to the request is received. The JSON object provided in the response is passed to the callback function. (Optional)
jsono	The state object to pass to the JSON callback function. You can use a state object to match a response with a specific call. This value is provided as the second parameter to the callback function provided in the JSONP parameter. (Optional)

There are a large number of search parameters that can be used with these API's. To keep things simple and easy to understand we will only look at the most common parameters. Additional, less commonly used parameters exist for many of these API's. Full documentation which contain additional examples on the Bing Maps REST services can be accessed here: <http://bit.ly/10JoNVW>

All response from the Bing Maps REST Services have a common structure. The response contains a number of properties, the main one of interest being the **ResourceSets** property. The **ResourceSets** property is an array of **Resource** objects. Each API has a different type of result class that it returns. These result classes inherit from the **Resource** class. Here is a simple list showing the hierarchy of these classes:

- Response
 - ResourceSets : ResourceSet[]
 - ResourceSet : Resource[]
 - Location
 - Route
 - ImageryMetadata
 - TrafficIncident
 - ElevationData
 - CompressedPointList
 - SeaLevelData

These classes have a number of additional properties. A complete set of class diagrams for the Bing Maps REST Service can be found in Appendix B. Note that the name of the properties in these class diagrams use managed code formatting which use camel casing and the first character capitalized. The JSON responses themselves start the property names with a small letter. Later in this chapter we will see how to access these API's from managed code from a library that uses these naming conventions.

Locations API

The location API provides you with geocoding and reverse-geocoding functionality. Geocoding is the process of determining a specific coordinate for an address or other geographic feature. Whereas reverse-geocoding, like its

name, does the reverse and is the process of determining the address of geographic feature information for a coordinate. All requests to the Location API use a common base URL:

`http://dev.virtualearth.net/REST/v1/Locations?[SearchParameters]&key=BingMapsKey`

Geocoding Addresses

One of the most common tasks carried out by Bing Maps users is geocoding an address. The data you want to geocode will likely be in one of two formats; structured address separated into parts, or a single line address. The following search parameters can be used to geocode a structured address.

Parameter	Description
addressLine	The street line of an address. (Optional)
locality	The city or town. (Optional)
postalCode	The postal or zip code. (Optional)
adminDistrict	The state or province. (Optional)
countryRegion	The ISO country code for the country. (Optional) A complete list of country ISO codes can be found in Appendix A.
includeNeighborhood	Specifies to include the neighborhood with the address information in the response when it is available. (Optional) <ul style="list-style-type: none"> 1: Include neighborhood information when available. 0: Do not include neighborhood information. (default)

Using these parameters the following address can be used to create a URL for geocoding:

Address Line: 1 Microsoft Way

City: Redmond

State: WA

Zip Code: 98052

Country: US

The URL to geocode this address look like this:

`http://dev.virtualearth.net/REST/v1/Locations?addressLine=1%20Microsoft%20Way&locality=Redmond&postalCode=98052&adminDistrict=WA&countryRegion=US&key=BingMapsKey`

Notice that the spaces in the address line are encoded to %20 characters. All string search parameters should be encoded. This ensures that special characters do not cause issues with the URL. If you opened this URL in a browser you will find the response contains the following information.

```
{
  "resourceSets": [
    {
      "estimatedTotal": 1,
      "resources": [
        {
          "bbox": [47.636267802296267, -122.13737510202641, 47.64399323743762, -122.12208859564548],
          "name": "1 Microsoft Way, Redmond, WA 98052",
          "point": {
            "type": "Point",
            "coordinates": [47.640130519866943, -122.12973184883595]
          },
          "address": {
            "addressLine": "1 Microsoft Way",

```

```

        "adminDistrict": "WA",
        "adminDistrict2": "King Co.",
        "countryRegion": "United States",
        "formattedAddress": "1 Microsoft Way, Redmond, WA 98052",
        "locality": "Redmond",
        "postalCode": "98052"
    },
    "confidence": "High",
    "entityType": "Address",
    "geocodePoints": [
    {
        "type": "Point",
        "coordinates": [47.640130519866943, -122.12973184883595],
        "calculationMethod": "InterpolationOffset",
        "usageTypes": ["Display"]
    },
    {
        "type": "Point",
        "coordinates": [47.640154659748077, -122.12978817522526],
        "calculationMethod": "Interpolation",
        "usageTypes": ["Route"]
    }
    ],
    "matchCodes": ["Good"]
}
}
}

```

Note that some unnecessary properties were removed from this response to make it less cluttered and easier to see the result.

Tip: You may find it easier to view the results as XML in a browser rather than as JSON. To do this simply change the output of the request to XML by adding the following to your URL: **&output=xml**

Geocoding a Single Line Address

In the past it was common to provide users with multiple textboxes where they had to separate all the parts of their address. In recent years it has become increasingly more popular to provide users with a single textbox where they can provide a single string query of their address. The following search parameters can be used to perform a single line address geocode.

Parameter	Description
query	A string that contains information about a location, such as an address or landmark name. (Required)
include	Specifies to include the parsed query string values or ISO 2 country code value in the response. The optional parameters are ciso2 and queryParse . When the ciso2 parameter is used the ISO 2 country code value is returned in the response as part of the address using the countryRegionIso2 field. When the queryParse parameter value is specified, the response shows how the query string was parsed into address values, such as addressLine , locality , adminDistrict , and postalCode . (Optional)
includeNeighborhood	Specifies to include the neighborhood with the address information in the response when it is available. (Optional) <ul style="list-style-type: none"> 1: Include neighborhood information when available. 0: Do not include neighborhood information. (default)
maxResults	Specifies the maximum number of locations to return in the response. The default value is 5. This can be an integer between 1 and 20. (Optional)

Using these parameters we can create a URL to geocode the address string "1 Microsoft Way, Redmond, WA".

`http://dev.virtualearth.net/REST/v1/Locations?query=1%20Microsoft%20Way,%20Redmond,%20WA&key=BingMapsKey`

If you open this URL in a browser it will return the exact same result that was returned in the previous section.

Geocoding Tips

For the most part the geocoding functionality does a good job of trying to determine the most likely result that matches your search. However there are a number of things you can do to help the geocoder return the most accurate results. Here are some useful tips when geocoding:

- Unless you are geocoding English addresses in the US you should specify a culture parameter to help ensure you get the most relevant results. By default the culture for all requests is set to "en-US".
- Whenever possible include a country in the query.
- Encode address and query parameters. This is especially important when working with non-English languages as special characters often will not work correctly. This is also important if you want to use an ampersand (&) in your query. By encoding an ampersand it will appear in the URL as "%26". Here are the methods that can be used in different programming languages to encode URL parameters.

Language	Method	Example
JavaScript	encodeURIComponent	encodeURIComponent(query/address)
C#/VB	Uri.EscapeDataString	Uri.EscapeDataString (query/address)

- The Bing Maps geocoder will attempt to find the closest match as possible to your query. In some cases it will not be able to find an exact match. This is where the match code parameter of the returned results becomes useful. The match code parameter is an array of values and can have any combination of the following three values; Good, Ambiguous, and UpHierarchy. If you are only interested in exact matches then keep a look out for UpHierarchy as this indicates that your exact query was not found but an upper level address value was found. For example, you attempt to geocode a postal code but instead the associated country is returned as the postal code was not found.

Reverse Geocoding Coordinates

Reverse geocoding allows you to get the address or location information for a specific coordinate. This is often used in apps that allow the user to place a pushpin on a map or select a point on a map as an end point to a route. The URL format for reverse geocoding is slightly different from what is used for geocoding as you can see here:

`http://dev.virtualearth.net/REST/v1/Locations/[Point]?[SearchParameters]&key=BingMapsKey`

The **Point** parameter in the URL is required and is a coordinate value represented as a string in the format "latitude,longitude". The following search parameters can be used to geocode an address.

Parameter	Description
include	Specifies to include the ISO 2 country code value in the response. The optional parameter is ciso2 . When the ciso2 parameter is used the ISO 2 country code value is returned in the response as part of the address using the countryRegionIso2 field. (Optional)
includeEntityTypes	Specifies the entity types that you want to return in the response. Only the types you specify will be returned. If the point cannot be mapped to the entity types you specify, no location information is returned in the response. (Optional) A comma separated list of entity types selected from the following options.

	<ul style="list-style-type: none"> • Address • Neighborhood • PopulatedPlace • Postcode1 • AdminDivision1 • AdminDivision2 • CountryRegion
includeNeighborhood	<p>Specifies to include the neighborhood with the address information in the response when it is available. (Optional)</p> <ul style="list-style-type: none"> • 1: Include neighborhood information when available. • 0: Do not include neighborhood information. (default)

Using these parameters we can create a URL to reverse geocode a coordinate that has a latitude value of 47.64054 and a longitude value of -122.12934.

<http://dev.virtualearth.net/REST/v1/Locations/47.64054,-122.12934&key=BingMapsKey>

If you open this URL in a browser it will return the exact same result that was returned in the Geocoding Addresses section as this is the coordinate for "1 Microsoft Way, Redmond, WA".

Reverse Geocoding Tip

When converting coordinate values into a string it is a good practice to reduce the number of significant digits of the numbers to 5 or 6 and to also use an invariant culture when converting to a string. This will ensure the point parameter of the URL is correctly formatted. Here is an example of how to do this:

JavaScript

```
var point = latitude.toFixed(5) + "," + longitude.toFixed(5);
```

C#

```
string point = string.Format("{0:N5},{1:N5}", latitude, longitude);
```

Visual Basic

```
Dim point As String = String.Format("{0:N5},{1:N5}", latitude, longitude)
```

Routes API

You can calculate Driving, Walking and Transit directions using the Routes API. Routes include information such as route instructions, travel time and distance or transit information. All requests to the Routes API use a common base URL:

[http://dev.virtualearth.net/REST/v1/Routes/\[TravelMode\]?\[RouteOptionParameters\]&key=BingMapsKey](http://dev.virtualearth.net/REST/v1/Routes/[TravelMode]?[RouteOptionParameters]&key=BingMapsKey)

The travel mode is an optional parameter that defaults to **Driving**. You can also set the travel mode to **Walking** or **Transit**. There are a large number of route option parameters that you can use to customize how your route is calculated. Here is a list of the most commonly used parameters:

Parameter	Description
waypoint.n	<p>Specifies two or more locations that define the route and that are in sequential order. (Optional)</p> <p>A short form version of this parameter is available: wp.n</p>

avoid	<p>A comma-separated list of values from the following list that limit the use of highways and toll roads in the route. (Optional)</p> <ul style="list-style-type: none"> • highways: Avoids the use of highways in the route. • tolls: Avoids the use of toll roads in the route. • minimizeHighways: Minimizes (tries to avoid) the use of highways in the route. • minimizeTolls: Minimizes (tries to avoid) the use of toll roads in the route. <p>If no values are specified, highways and tolls are allowed in the route.</p>
optimize	<p>Specifies how to optimize the route.</p> <ul style="list-style-type: none"> • distance: The route is calculated to minimize the distance. Traffic information is not used. • time [default]: The route is calculated to minimize the time. Traffic information is not used. • timeWithTraffic: The route is calculated to minimize the time and uses current traffic information. <p>Note: This is not a traveling sales men type optimizing. This will not reorder the waypoints to create the shortest route between all points. If you require that functionality take a look at the Bing Maps Trip Optimizer code sample: http://bit.ly/10K6mQR</p>
routePathOutput	<p>Specifies whether the response should include the route's path coordinates. (Optional)</p> <ul style="list-style-type: none"> • Points: A list of Point values for the route's path is provided in the response. • None [default]: No information about the route's path is provided in the response.
distanceUnit	<p>The units to use for distance in the response. (Optional)</p> <ul style="list-style-type: none"> • Mile or mi • Kilometer or km [default]

Understanding Waypoints

A route is defined by a set of waypoints. There are several different types of waypoints you can specify.

Waypoint Type	Example
Point	47.610,-122.107
Landmark	Eiffel%20Tower
Address	1%20Microsoft%20Way,%20Redmond,%20WA

The index (n value) for the set of waypoints is an integer starting with 0 or 1. The waypoint index values must be sequential and must always increment by 1. A route can have a maximum of 25 waypoints. The waypoint part of a query to calculate a route from London to Paris would have the following format:

&wp.1=London&wp.2=Paris

Calculating Driving & Walking Directions

Using the route option parameters we can easily create a URL to calculate a driving or walking route between two locations. The following URL could be used to calculate a driving route that goes between Seattle and Redmond and also avoids toll roads:

<http://dev.virtualearth.net/REST/V1/Routes/Driving?wp.0=Seattle&wp.1=Redmond&avoid=tolls&key=BingMapsKey>

Using the URL you will be able to get the full driving instructions for this route. These instructions will include the driving time and distance along with the turn by turn directions. If you want to display the route on a map you would

need the coordinate information that makes up the route path. This is not returned by default as it makes the response significantly larger. If you wanted to retrieve the coordinates for the route path you can add the `routePathOutput` parameter to the URL like so:

```
http://dev.virtualearth.net/REST/V1/Routes/Driving?wp.0=Seattle&wp.1=Redmond&avoid=tolls&routePathOutput=Points&key=BingMapsKey
```

Driving directions are the most common type of directions generated by Bing Maps users but it is sometimes useful to generate walking directions. Walking directions ignore the rules of one way streets and sometimes will use walking paths if available. When driving we typically want to calculate routes that take the least amount of time, however when walking we will generally prefer to take the shortest route. We can use the `optimize` parameter in the URL to indicate this. The following URL is an example of how to optimize walking directions by distance when walking from the Eiffel Tower to the Louvre Museum in Paris.

```
http://dev.virtualearth.net/REST/V1/Routes/Walking?wp.0=Eiffel%20Tower&wp.1=Louvre%20Museum&optimize=distance&key=BingMapsKey
```

In addition to calculating a route between two locations, Bing Maps can also calculate a route of up to 25 waypoints in a single request. With this in mind let's say you are planning a trip from Phoenix to Los Angeles but to make things a bit more interesting let's add a stopover in Las Vegas. This sounds like a perfectly logical trip to take and you can calculate the directions between these 3 waypoints using the following URL:

```
http://dev.virtualearth.net/REST/V1/Routes/Driving?wp.0=Phoenix&wp.1=Las%20Vegas&wp.3=Los%20Angeles&key=BingMapsKey
```

Responses from the route service are very large and would easily take up to 10 pages of this book and not add very much value. In addition to this the Routes API can also calculate transit directions however there are a much longer list of route option parameters available for that type of route which I have not included in this book. You can find example responses and addition details on transit routing in the MSDN documentation for the Routes API here: <http://bit.ly/10mc4cY>

Routing Tips

Many of the tips in the Location API apply for the routing service as well. For instance being sure to encode your address waypoints. However, don't encode coordinate based waypoints. If using Bing Maps in areas where geocoding coverage is limited consider allowing the user to select their start and end point on the map via a click or by dragging a pushpin. This will allow you to pass in coordinates for your end points rather than an address. The routing engine is capable of calculating routes anywhere there is road data, even if there is no geocoding coverage in the area.

Geographic coverage information Bing Maps can be found here: <http://bit.ly/183rXMT>

Imagery API

The Imagery API provides you with the ability to generate images of maps and to also retrieve metadata about the imagery. Using static imagery of maps is a great option if you only need to show something on a map and it doesn't need to be interactive. An image of a map uses a lot less resources than loading the full Bing Maps control would. Static maps are also an excellent way to share maps. There are a couple of different URL formats that can be used to generate a static map image.

```
http://dev.virtualearth.net/REST/v1/Imagery/Map/[ImagerySet]?[ImageOptionParameters]&key=BingMapsKey
```

Get a map that is centered at a specified point.

[http://dev.virtualearth.net/REST/v1/Imagery/Map/\[ImagerySet\]/\[CenterPoint\]/\[ZoomLevel\]?\[ImageOptionParameters\]&key=BingMapsKey](http://dev.virtualearth.net/REST/v1/Imagery/Map/[ImagerySet]/[CenterPoint]/[ZoomLevel]?[ImageOptionParameters]&key=BingMapsKey)

Get a map that is based on a query.

[http://dev.virtualearth.net/REST/v1/Imagery/Map/\[ImagerySet\]/\[Query\]?\[ImageOptionParameters\]&key=BingMapsKey](http://dev.virtualearth.net/REST/v1/Imagery/Map/[ImagerySet]/[Query]?[ImageOptionParameters]&key=BingMapsKey)

Get a map that displays a route.

[http://dev.virtualearth.net/REST/v1/Imagery/Map/\[ImagerySet\]/Routes/\[TravelMode\]?\[ImageOptionParameters\]&key=BingMapsKey](http://dev.virtualearth.net/REST/v1/Imagery/Map/[ImagerySet]/Routes/[TravelMode]?[ImageOptionParameters]&key=BingMapsKey)

Get a map that is centered over a portion of a route.

[http://dev.virtualearth.net/REST/v1/Imagery/Map/\[ImagerySet\]/\[CenterPoint\]/\[ZoomLevel\]/Routes/\[TravelMode\]?\[ImageOptionParameters\]&key=BingMapsKey](http://dev.virtualearth.net/REST/v1/Imagery/Map/[ImagerySet]/[CenterPoint]/[ZoomLevel]/Routes/[TravelMode]?[ImageOptionParameters]&key=BingMapsKey)

The **imagery set** is the type of map style you want the map to have. The **ImagerySet** parameter can have one of the following values:



Aerial



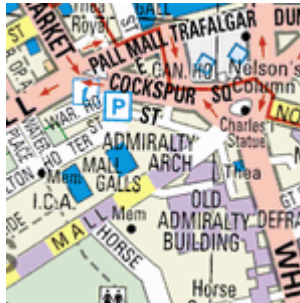
AerialWithLabels



Road



OrdnanceSurvey (UK only)



CollinsBart (London UK only)

The **CenterPoint** value of the URL is used to center the map over a specific coordinate. The **ZoomLevel** is an integer value between 1 and 21 that indicates the level at which the map image should be zoomed into. The **Query** value can be used instead of a center point and zoom level. A query can be an address or a landmark. The following are the image option parameters that can be used with the Imagery API.

Parameter	Description
declutterPins	Specifies whether to change the display of overlapping pushpins so that they display separately on a map. (Optional) <ul style="list-style-type: none"> 1: Declutter pushpin icons. 0: Do not declutter pushpin icons. (default)

format	The image format to use for the static map. By default the best file format for a particular map style is used. (Optional) <ul style="list-style-type: none"> • gif: GIF image format. • jpeg: JPEG image format. JPEG format is the default for Road, Aerial and AerialWithLabels imagery. • png: PNG image format. PNG is the default format for CollinsBart and OrdnanceSurvey imagery.
mapArea	The geographic area to display on the map. Required when a center point or set of route points are not specified. This is a comma separated set of numbers using the following format: "South Latitude,West Longitude,North Latitude,East Longitude"
mapLayer	A display layer that renders on top of the imagery set. The only value for this parameter is TrafficFlow . (Optional) Example: mapLayer=TrafficFlow
mapSize	A string that contains a width and a height separated by a comma. The width must be between 80 and 900 pixels and the height must be between 80 and 834 pixels. The default map size for static maps is 350 pixels by 350 pixels. (Optional)
pushpin	A series of values that include a Point value (latitude and longitude) with options to add a label of up to three (3) characters and to specify an icon style. You can specify up to 18 pushpins within a URL and 100 if you use the HTTP POST method and specify the pushpins in the body of the request. (Optional) A short form version of this parameter is available: pp
mapMetadata	Specifies whether to return metadata for the static map instead of the image. The static map metadata includes the size of the static map and the placement and size of the pushpins on the static map. (Optional) <ul style="list-style-type: none"> • 1: Return metadata for the specific image. An image is not returned. • 0: Do not return metadata. (default) When you request metadata, the response returns metadata for the map instead of the map image.

In addition to these parameters the Imagery API also supports the route option parameters that we covered in the previous section. Using the route option parameters with the imagery API allows you to generate map images that have routes drawn on them.

Generating Map Images

You can easily create map images using the different parameters available through the Bing Maps REST Imagery service. For instance, the following URL can be used to generate a road map image centered over the Eiffel Tower that has a width of 400 pixels and a height of 200 pixels.

```
http://dev.virtualearth.net/REST/V1/Imagery/Map/Road/Eiffel%20Tower?mapSize=400,200&key=BingMap
sKey
```

This request generates the following image:



In addition to using text queries as the point to center the map on you can also use the coordinates that are written as text using the format of "latitude,longitude". The following URL generates an aerial map image that is centered over coordinates (47.62,-122.349) which is the approximate location of the Space Needle in Seattle. The map is zoomed into zoom level 17 and will be returned using the default map size of 350 pixels by 350 pixels.

<http://dev.virtualearth.net/REST/v1/Imagery/Map/Aerial/47.62,-122.349/17?key=BingMapsKey>

This request generates the following image:



Drawing Maps with Pushpins

Pushpins identify locations on a map. In addition to specifying where to place a pushpin you can also specify an icon style and a label for a pushpin. Pushpins are defined using the following format:

`&pushpin=latitude,longitude;iconStyle;label`

When creating a pushpin you must always specify a location, however the icon style and label are optional. The icon style is an integer value that relates to a predefined icon type. A complete list of valid icon styles can be found in Appendix C. This table shows the four different ways to customize a pushpin in the Imagery API.

Description	Syntax
Specify a pushpin location.	latitude,longitude Example: pushpin=47.620548,-122.34874
Specify a pushpin location and an icon style.	latitude,longitude;iconStyle Example: pushpin=47.620548,-122.34874;5
Specify a pushpin location and a label.	latitude,longitude;;label Note: The order of the syntax requires that if you specify a label without specifying an icon style, you must preserve the syntax and put two semi-colons between the coordinates and the label. Example: pushpin=47.620548,-122.34874;;P10
Specify a pushpin location, an icon style, and a label. A label can have up to three (3) characters.	latitude,longitude;iconStyle;label Example: pushpin=47.620548,-122.34874;5;P10

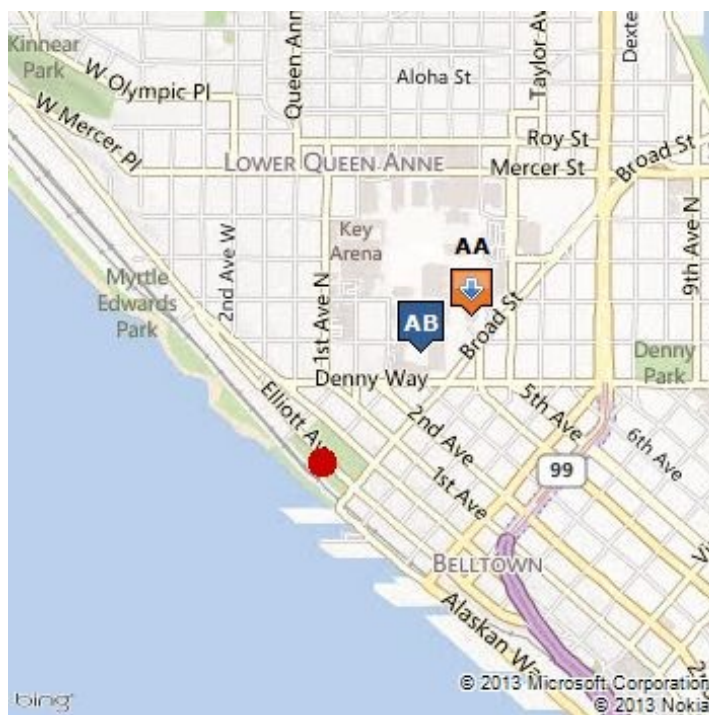
In addition to specifying pushpins using the “pushpin” parameter you can also use the short form version of this parameter which is “pp”. You can specify up to 18 pushpins when generating static images using a URL and loading via a HTTP GET request. However, if you use HTTP POST, you can specify up to 100 pushpins by inserting them into the request body. For more information on using the POST method see the documentation here:

<http://bit.ly/13MQP4R>

Using these parameter we can create a map with road imagery and pushpins on the Space Needle, the Pacific Science Center, and the Olympic Sculpture Park in Seattle. The center point of the map is set to 47.61904 degrees latitude and -122.35384 degrees longitude and the zoom level is set to 14.

`http://dev.virtualearth.net/REST/v1/Imagery/Map/Road/47.61904,-122.35384/14?pp=47.620495,-122.34931;21;AA&pp=47.61938,-122.35148;;AB&pp=47.616295,-122.3556;22&key=BingMapsKey`

This request generates the following image:

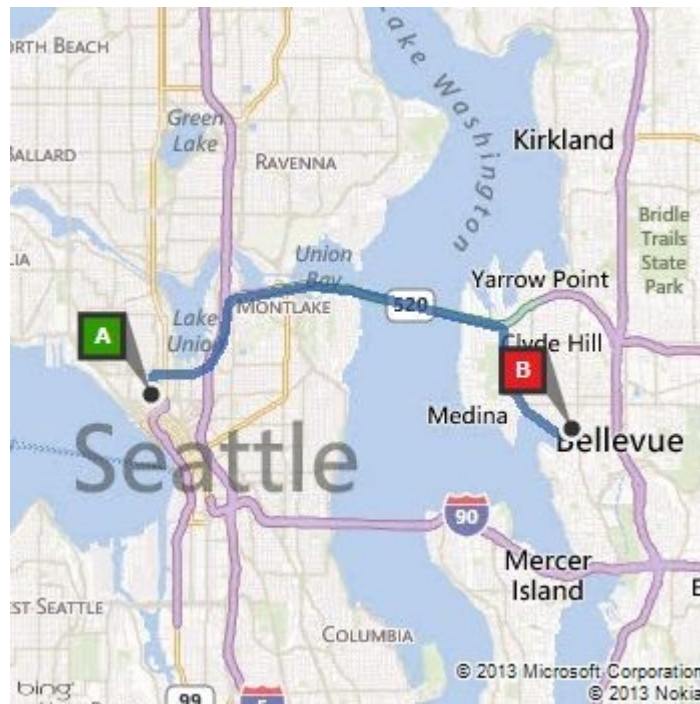


Drawing Maps with Routes

In addition to generating maps with pushpins you can also generate maps that have routes drawn on them. The following example shows how to get a map with road imagery that displays a driving route from the Space Needle in Seattle, Washington to Bellevue Downtown Park in Bellevue, Washington.

<http://dev.virtualearth.net/REST/V1/Imagery/Map/Road/Routes/Driving?wp.0=Space%20Needle&wp.1=Bellevue%20Downtown%20Park&key=BingMapsKey>

This URL generates a nice overview of the route from start to finish.



Viewing the whole route is nice but it might be useful to also have a map that is zoomed into the start of the route so that you have better idea of which roads you will be driving on. You can zoom into any coordinate on the map but if you are using the REST services to also generate the route instructions you can use the coordinates from that response to get the points along the route. The start of the previous route image is the Space Needle which we know to have a coordinate of (47.62,-122.349). We can easily take the previous route URL and insert this center point and a zoom level in between the image type and Routes parameter in the URL like so:

<http://dev.virtualearth.net/REST/V1/Imagery/Map/Road/47.62,-122.349/15/Routes/Driving?wp.0=Space%20Needle&wp.1=Bellevue%20Downtown%20Park&key=BingMapsKey>

This URL generates the following map image:



Making Use of Sessions

This section will go through and highlight many useful tips to help you make the most of your Bing Maps sessions and transactions. This can have a huge impact on the number of billable transactions that get generated by your application and as a result make a big difference in terms of licensing. To start off, there are a couple of definitions you should know.

- **Bing Maps Session:** A Bing Maps session occurs when one of the map controls is loaded. Any transactions occurred against the Bing Maps services, while within a session is non-billable. Note this requires the application to properly use Bing Maps Keys.
- **Bing Maps Transaction:** A Bing Maps transaction occurs any time a service request is made. For example, some of the more common services used that incur transactions are: Bing Maps Geocoding, Routing, and Imagery service, and Bing Spatial Data Service Query's.
- **Bing Maps Key:** A Bing Maps Key is a unique string that is used to authenticate a user's Bing Maps application or service request. This is the primary method used for tracking usage of the Bing Maps API's.

Complete documentation explaining all the different ways sessions and transactions are incurred can be found here <http://bit.ly/130UwmS>

Many of the Bing Maps API's have a method for getting the credentials from the map after you have loaded it using a valid Bing Maps key. One often overlooked feature is that, by getting the credentials from the map, you do not get back your original Bing Maps key. Instead, you get a special session key which you can use as a Bing Maps key to make requests to the Bing Maps services. By doing this, all transactions occurred by this session key will be non-billable. Many developers overlook this feature and opt to simply use their original Bing Maps key, not knowing that they are actually incurring more billable transactions than they need to.

Implementing Session Keys

Implementing session keys is pretty straight forward. You simply use a session key instead of a Bing Maps Key. When using the REST services, simply use your session key with the **key** parameter of your requests. The following code shows how to generate a session key from the map.

JavaScript

```
var map, sessionKey;

map = new Microsoft.Maps.Map(document.getElementById("MyMap"), {
    credentials: "YOUR_BING_MAPS_KEY"
});

//Get session key from map after it is loaded
map.getCredentials(function (c) {
    sessionKey = c;
});
```

C#

```
private string sessionKey;

//Get session key from map after it is loaded
MyMap.Loaded += async (s, e) =>
{
    try
    {
        sessionKey = await MyMap.GetSessionIdAsync();
    }
    catch { }
};
```

Visual Basic

```
Private sessionKey As String

'Get session key from map after it is loaded
AddHandler MyMap.Loaded, Async Sub(s, a)
    Try
        sessionKey = Await MyMap.GetSessionIdAsync()
    Catch
    End Try
End Sub
```

Design Tips to Optimize Sessions

Here are a few tips to maximize your use of sessions:

1. Generate a session key right after the map loads and store it in a global variable inside your app. This will save you time later in your application when you need to use it.
2. If you have a long running application where you expect the user to be using the app for hours on end then generate the session key for each request to ensure the key doesn't time out. Alternatively setup a timer to update the session key every hour or so.
3. If possible create the map on the main page of the app (default.html/MainPage.xaml) and position it relative to content on other pages. This will allow you to load the map once and reuse it elsewhere in the app.

Sharing Maps using the Share Charm

Depending on the type of application you are creating, you may find it useful to be able to share a map with someone. Whether it's a map of a single location or a map with a route on it, this can easily be accomplished in a Windows Store app. In this code sample we are going to take a look at two different ways you can share a map in a Windows Store app.

Sharing things is easy in Windows Store apps when using the Share charm. We can easily specify text or HTML that we want to share inside a message. There are two ways we can do share a map. The first is to use the Bing Maps REST Imagery service and generate an image of a map. The second method is to make use of the Bing Maps consumer site (<http://bing.com/maps>) and build a URL.

Creating a URL to Bing Maps

The Bing Maps consumer site exposes a number of URL query parameters which can be used to load a custom map view. These parameters are very similar to the Bing Maps REST Imagery service. You can customize the URL to display specific search results, driving directions, or items in your "My Places" folder.

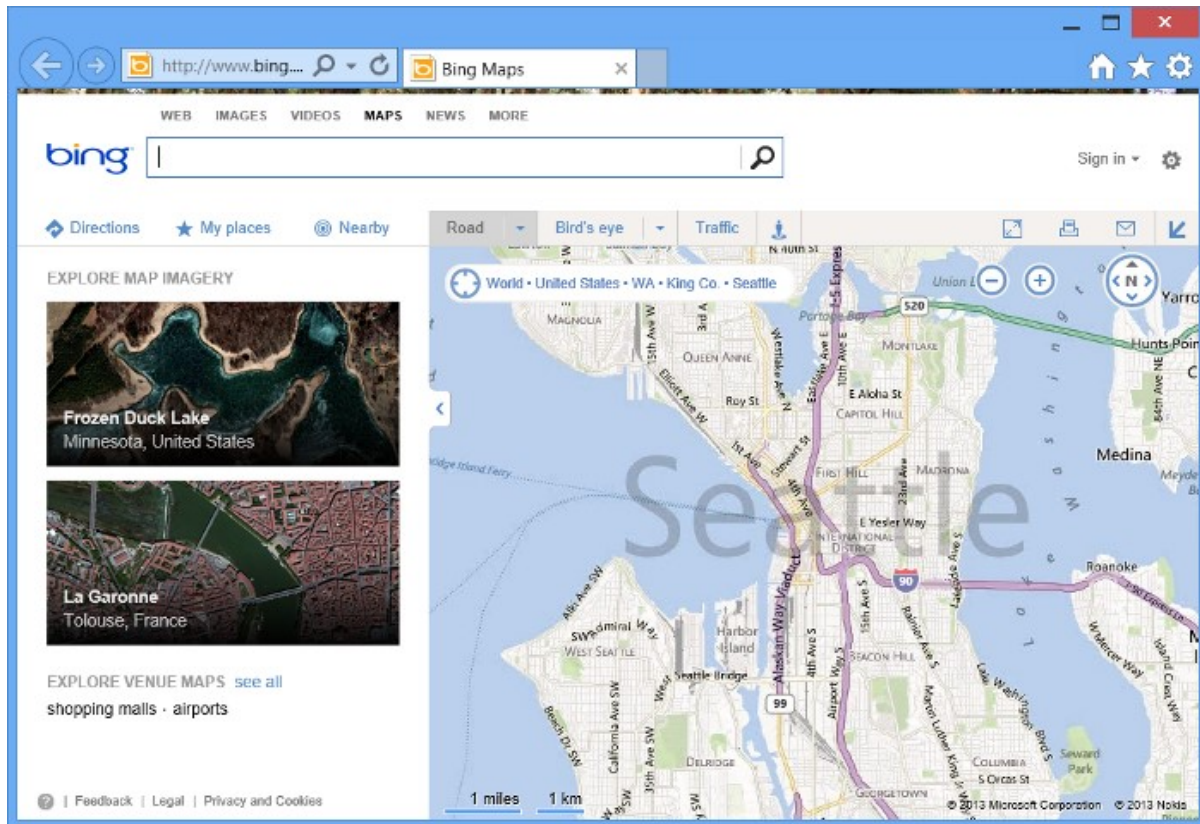
To create a URL to the Bing Maps consumer site, start with the base URL and then use parameters to specify the location and options such as zoom level, map view, search panels, and more. The base address you should use is <http://bing.com/maps/default.aspx>. The following is a list of some of the basic parameters you can use:

Parameter	Description
cp	Defines where the center of the map should be. Use the following format for the cp parameter: Latitude~Longitude
lvl	Defines the zoom level of the map. Valid values are 1-19. This parameter is ignored if you don't include the cp parameter.
style	Defines the map view. Valid values for this parameter include: a : Display an aerial view of the map. r : Display a road view of the map. h : Display an aerial view of the map with labels. o : Use this value to display a bird's eye (oblique) view of the map. b : Display a bird's eye (oblique) with labels view of the map. u : Automatic
trfc	Specifies whether traffic information is included on the map. Omitting the trfc parameter produces the same results as trfc=0.

You can find full documentation on creating URLs for the Bing Maps consumer site on the Bing Help website here: <http://bit.ly/1eLLPW0>

Using this information we can create the following URL that opens Bing Maps with the map centered on a specific location (Seattle - 47.60356,-122.32943) with a zoom level of 12, and the map view set to the road map view:
<http://bing.com/maps/default.aspx?cp=47.60356~-122.32943&lvl=12&style=r>

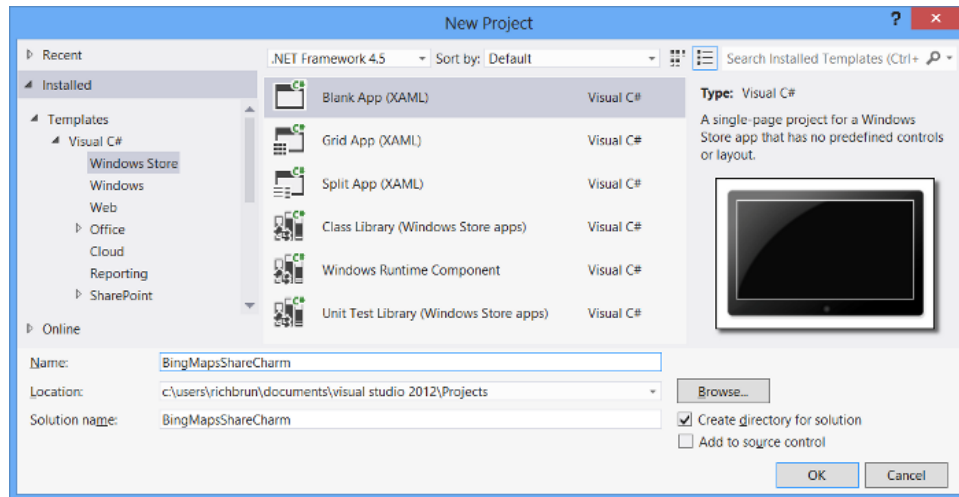
Opening this URL in a browser will load up the Bing Maps consumer site centered over Seattle.



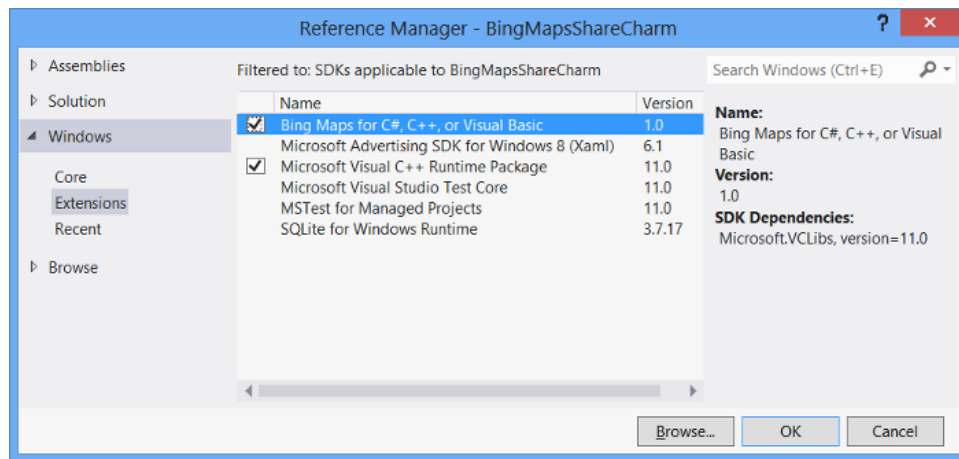
Sharing from a Windows Store App

Now that we understand the basics of generating a static map image from earlier in this chapter and how to generate a link to the Bing Maps consumer site, we can use this information to create a Windows Store app that allows us to share maps through the Share charm. For this app we will create a simple application that uses the Bing Maps Windows Store control. When the Share charm is activated it will use the center point and zoom level of the map to generate the map image and URL to the Bing Maps consumer site. In addition to this, we will also add a button to the app that will launch the share charm. You may find this useful in the future if you have a list of results and you want to let the user select a single result and share it with a single touch.

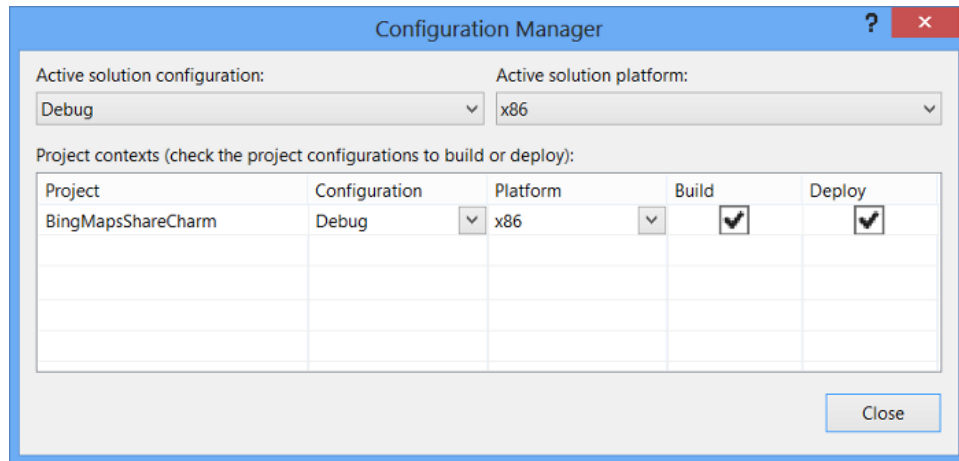
To get started let's open up Visual Studios and create a new project in your preferred language: JavaScript, C# or Visual Basic. Select the **Blank App** template and call the application **BingMapsShareCharm** and press OK.



Add a reference to the Bing Maps SDK. To do this, right click on the **References** folder and press **Add Reference**. Select **Windows** -> **Extensions** select **Bing Maps for C#, C++ and Visual Basic** or **Bing Maps for JavaScript**. If you do not see this option ensure that you have installed the Bing Maps SDK for Windows Store apps. While you are here, if using C# or Visual Basic, add a reference to the **Microsoft Visual C++ Runtime Package** as this is required by the Bing Maps SDK.



If you are using C# or Visual Basic you may notice that there is a little yellow indicator on the references that you just added. The reason for this is that in the C++ runtime package you have to set the **Active solution platform** in Visual Studio to one of the following options; **ARM**, **x86** or **x64**. To do this, right click on the Solution folder and select **Properties**. Then go to **Configuration Properties -> Configuration**. Find your project and under the **Platform** column set the target platform to x86. Press OK and the yellow indicator should disappear from our references.



If you are using JavaScript right click on the **js** folder and select **Add -> New Item**. Create a new JavaScript file called **ShareMap.js**. We will put all our JavaScript for this application in there to keep things clean. Now open up the **default.html** file and update the HTML to the following:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>BingMapsShareCharm</title>

  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.1.0/css/ui-dark.css" rel="stylesheet" />
  <script src="//Microsoft.WinJS.1.0/js/base.js"></script>
  <script src="//Microsoft.WinJS.1.0/js/ui.js"></script>

  <!-- BingMapsShareCharm references -->
  <link href="/css/default.css" rel="stylesheet" />
  <script src="/js/default.js"></script>

  <!-- Bing Map Control references -->
  <script type="text/javascript" src="ms-appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>

  <!-- Our Bing Maps JavaScript Code -->
  <script src="/js/ShareMap.js"></script>
</head>
<body>
  <div id="MyMap"></div>
  <button id="ShareBtn" style="position:absolute;top:0;left:400px;background-color:#000;">Share Location</button>
</body>
</html>
```

If you are using C# or Visual Basic open the **MainPage.xaml** file and add update the XAML with the following:

```
<Page
  x:Class="BingMapsShareCharm.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:BingMapsShareCharm"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:m="using:Bing.Maps"
```

```

mc:Ignorable="d">

<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <m:Map Name="MyMap" Credentials="YOUR_BING_MAPS_KEY"/>

    <StackPanel Margin="10" Height="40" VerticalAlignment="Top"
HorizontalAlignment="Center" Background="Black">
        <Button Content="Share Location" Tapped="ShareLocation_Tapped"/>
    </StackPanel>
</Grid>
</Page>

```

Next we can add the functionality to power the application. Regardless of the language you are using you will have to do the following tasks:

1. Get a reference to the share charm for the current view using the **DataTransferManager**.
2. Add an event handler when the user navigates to the app that triggers when data is requested by the share charm. Remove this event handler when the user leaves the app.
3. Since we are using Bing Maps generate a session key from the map after the map has loaded. A session key is a special Bing Maps key that marks requests to the REST services as non-billable.
4. When a custom share button is tapped show the UI for the share charm.
5. Create an event handler for when the share charm is launched. To share an image through the share charm, generate some HTML with the layout you prefer. Add an image tag for where you want the image to go and provide it with a temporary URL. You can then load the image as a stream to the request and specify the temporary URL the image is meant for. In the HTML you can also add an anchor tag that links to the Bing Maps consumer site with the same map, but interactive.

If you are using JavaScript you have one more task and that is to load the Bing Maps control. Open the **ShareMap.js** file and add the following code to it.

```

(function () {
    var app = WinJS.Application;
    var map, sessionKey, dataTransferManager;

    app.onready = function (args) {
        //Get the DataTransferManager object associated with current window
        dataTransferManager =
        Windows.ApplicationModel.DataTransfer.DataTransferManager.getForCurrentView();

        //Register the share handler event
        dataTransferManager.addEventListener("datarequested", ShareHandler);
    };

    app.onunload = function (args) {
        //Unregister the share handler event
        dataTransferManager.removeEventListener("datarequested", ShareHandler);
    };

    function initialize() {
        Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: GetMap });

        document.getElementById("ShareBtn").addEventListener("click", function (e) {
            Windows.ApplicationModel.DataTransfer.DataTransferManager.showShareUI();
        }, true);
    }

    function GetMap() {
        map = new Microsoft.Maps.Map(document.getElementById("MyMap"), {

```

```

        credentials: "YOUR_BING_MAPS_KEY"
    });

    map.getCredentials(function(c){
        sessionKey = c;
    });
}

function ShareHandler(e){
    var mapCenter = map.getCenter();
    var lat = mapCenter.latitude.toFixed(5);
    var lon = mapCenter.longitude.toFixed(5);

    //A temporary image URL that is used for referencing the created map image.
    var localImage = "ms-appx:///images/map.png";

    //Use the Bing Maps REST Services to retrieve a static map image.
    var mapImageUrl = 'http://dev.virtualsearth.net/REST/v1/Imagery/Map/Road/' + lat +
    ',' + lon + '/' + map.getZoom() + '?mapSize=400,300&key=' + sessionKey;

    //Create URL to the Bing Maps consumer site
    var bingMapsUrl = 'http://bing.com/maps/default.aspx?v=2&cp=' + lat + '~' + lon +
    '&lvl=' + map.getZoom();

    //Handle the Share Charm request and insert HTML content.
    var request = e.request;
    request.data.properties.title = 'Share Map';
    request.data.properties.description = 'Share an image of the current map.';

    var html = "Map of: " + lat + ", " + lon + "<br/><br/><img src='" + localImage +
    "'/><br/><a href='" + + "'>View on Bing Maps</a>";

    request.data.setHtmlFormat(Windows.ApplicationModel.DataTransfer.HtmlFormatHelper.createHtml
    Format(html));

    //Load the map image into the email as a stream.
    var streamRef =
    Windows.Storage.Streams.RandomAccessStreamReference.createFromUri(new
    Windows.Foundation.Uri(mapImageUrl));
    request.data.resourceMap[localImage] = streamRef;
}

document.addEventListener("DOMContentLoaded", initialize, false);
})();

```

Be sure to add your Bing Maps key into the **GetMap** method. Open the **MainPage.xaml.cs** file and update it with the following code.

C#

```

using System;
using Windows.ApplicationModel.DataTransfer;
using Windows.Storage.Streams;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Navigation;

namespace BingMapsShareCharm
{
    public sealed partial class MainPage : Page

```

```

{
    /// <summary>
    /// A reference to the DataTransferManager for the Share charm.
    /// </summary>
    private DataTransferManager dataTransferManager;

    private string sessionKey;

    public MainPage()
    {
        this.InitializeComponent();

        //Get the DataTransferManager object associated with current window
        dataTransferManager = DataTransferManager.GetForCurrentView();

        MyMap.Loaded += MyMap_Loaded;
    }

    private async void MyMap_Loaded(object sender, Windows.UI.Xaml.RoutedEventArgs e)
    {
        try
        {
            sessionKey = await MyMap.GetSessionIdAsync();
        }
        catch { }
    }

    protected override void OnNavigatedTo(NavigationEventArgs e)
    {
        //Register the share handler event
        dataTransferManager.DataRequested += ShareHandler;
    }

    protected override void OnNavigatedFrom(NavigationEventArgs e)
    {
        //Unregister the share handler event
        dataTransferManager.DataRequested -= ShareHandler;
    }

    private void ShareLocation_Tapped(object sender, TappedRoutedEventArgs e)
    {
        //Show the charm bar with Share option opened
        DataTransferManager.ShowShareUI();
    }

    private void ShareHandler(DataTransferManager sender, DataRequestedEventArgs e)
    {
        //A temporary image URL that is used for referencing the created map image.
        string localImage = "ms-appx:///images/map.png";

        //Use the Bing Maps REST Services to retrieve a static map image of the selected
        location.
        string mapImageUrl =
        string.Format("http://dev.virtualearth.net/REST/v1/Imagery/Map/Road/{0:N5},{1:N5}/{2}?mapSize=400,300&key={3}",
            MyMap.Center.Latitude, MyMap.Center.Longitude, Math.Round(MyMap.ZoomLevel),
            sessionKey);

        //Create URL to the Bing Maps consumer site
        string bingMapsUrl =
        string.Format("http://bing.com/maps/default.aspx?v=2&cp={0:N5}~{1:N5}&lvl={2}",

```



```

        MyMap.Center.Latitude, MyMap.Center.Longitude, Math.Round(MyMap.ZoomLevel));

        //Handle the Share Charm request and insert HTML content.
        DataRequest request = e.Request;
        request.Data.Properties.Title = "Share Map";
        request.Data.Properties.Description = "Share an image of the current map.";

        request.Data.SetHtmlFormat(HtmlFormatHelper.CreateHtmlFormat(
            string.Format("Map of: {0:N5}, {1:N5}<br/><br/><img src='{2}'/><br/><a
href='{3}'>View on Bing Maps</a>",
                MyMap.Center.Latitude, MyMap.Center.Longitude, localImage,
                bingMapsUrl)));

        //Load the map image into the email as a stream.
        RandomAccessStreamReference streamRef =
        RandomAccessStreamReference.CreateFromUri(new Uri(mapImageUrl));
        request.Data.ResourceMap[localImage] = streamRef;
    }
}
}

```

Visual Basic

```

Imports Windows.ApplicationModel.DataTransfer
Imports Windows.Storage.Streams
Imports Windows.UI.Xaml.Controls
Imports Windows.UI.Xaml.Input
Imports Windows.UI.Xaml.Navigation

Partial Public NotInheritable Class MainPage
    Inherits Page
    ''' <summary>
    ''' A reference to the DataTransferManager for the Share charm.
    ''' </summary>
    Private dataTransferManager As DataTransferManager

    Private sessionKey As String

    Public Sub New()
        Me.InitializeComponent()

        'Get the DataTransferManager object associated with current window
        dataTransferManager = dataTransferManager.GetForCurrentView()

        AddHandler MyMap.Loaded, AddressOf MyMap_Loaded
    End Sub

    Protected Async Sub MyMap_Loaded(sender As Object, e As Windows.UI.Xaml.RoutedEventArgs)
        Try
            sessionKey = Await MyMap.GetSessionIdAsync()
        Catch
        End Try
    End Sub

    Protected Overrides Sub OnNavigatedTo(e As NavigationEventArgs)
        'Register the share handler event
        AddHandler dataTransferManager.DataRequested, AddressOf ShareHandler
    End Sub

    Protected Overrides Sub OnNavigatedFrom(e As NavigationEventArgs)

```

```

        'Unregister the share handler event
        RemoveHandler dataTransferManager.DataRequested, AddressOf ShareHandler
    End Sub

    Private Sub ShareLocation_Tapped(sender As Object, e As TappedRoutedEventArgs)
        'Show the charm bar with Share option opened
        dataTransferManager.ShowShareUI()
    End Sub

    Private Sub ShareHandler(sender As DataTransferManager, e As DataRequestedEventArgs)
        'A temporary image URL that is used for referencing the created map image.
        Dim localImage As String = "ms-appx:///images/map.png"

        'Use the Bing Maps REST Services to retrieve a static map image of the selected
        location.
        Dim mapImageUrl As String =
        String.Format("http://dev.virtualearth.net/REST/v1/Imagery/Map/Road/{0:N5},{1:N5}/{2}?mapSize=400,300&key={3}", MyMap.Center.Latitude, MyMap.Center.Longitude,
        Math.Round(MyMap.ZoomLevel), sessionKey)

        'Create URL to the Bing Maps consumer site
        Dim bingMapsUrl As String =
        String.Format("http://bing.com/maps/default.aspx?v=2&cp={0:N5}~{1:N5}&lvl={2}",
        MyMap.Center.Latitude, MyMap.Center.Longitude, Math.Round(MyMap.ZoomLevel))

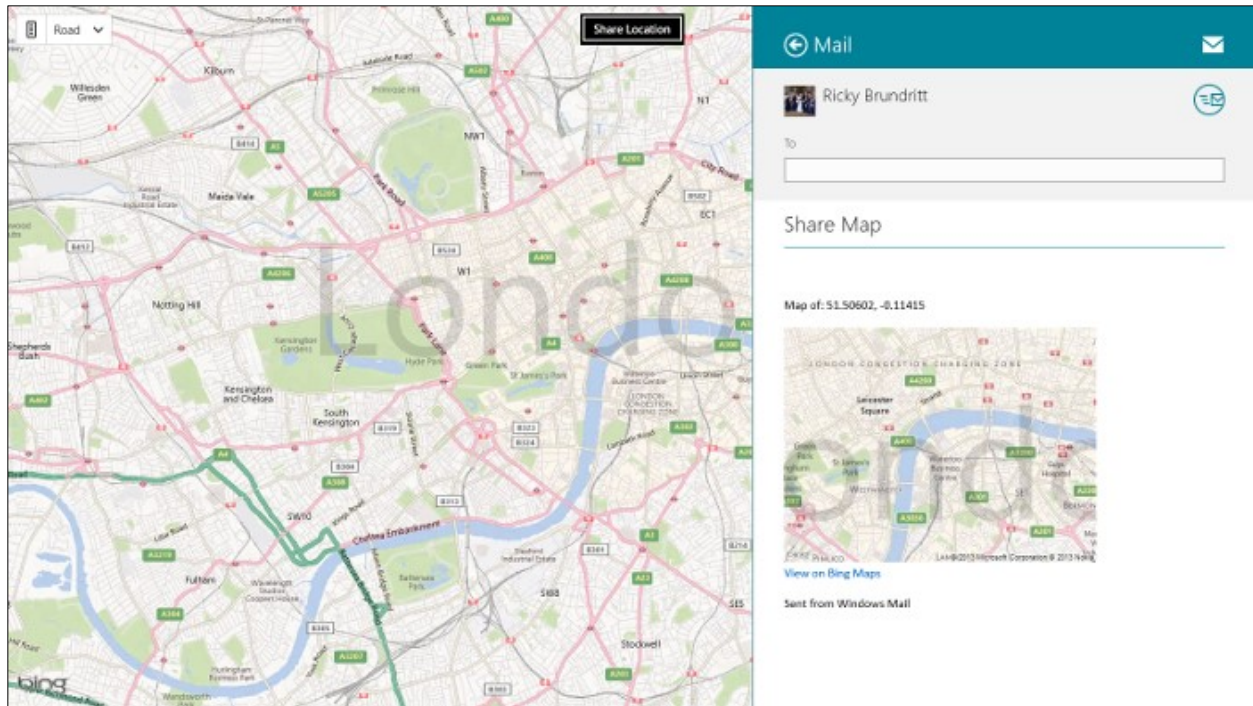
        'Handle the Share Charm request and insert HTML content.
        Dim request As DataRequest = e.Request
        request.Data.Properties.Title = "Share Map"
        request.Data.Properties.Description = "Share an image of the current map."

        request.Data.SetHtmlFormat(HtmlFormatHelper.CreateHtmlFormat(String.Format("Map of:
        {0:N5}, {1:N5}<br/><br/><img src='{2}'><br/><a href='{3}'>View on Bing Maps</a>",
        MyMap.Center.Latitude, MyMap.Center.Longitude, localImage, bingMapsUrl)))

        'Load the map image into the email as a stream.
        Dim streamRef As RandomAccessStreamReference =
        RandomAccessStreamReference.CreateFromUri(New Uri(mapImageUrl))
        request.Data.ResourceMap(localImage) = streamRef
    End Sub
End Class

```

Now run your application, and zoom in to a location. Press the share button you created or the Share charm button from the right flyout menu and select the Mail option. You should end up with an email with a map in it. Here is a screenshot of the app.



Implementing the Bing Maps REST Services

The Bing Maps REST services are accessed by taking a properly formatted URL and making a HTTP GET request against it. The Bing Maps REST services can return responses in XML or JSON format, the default is JSON. A JSON response is typically 25%-40% smaller than an XML response, and will take less time for the client application to download. We will only focus on the JSON responses in this book.

Some features in the Bing Maps REST Services also support HTTP Post requests however these are less commonly used and will not be covered in this section. Also note that for images you can simply pass the Bing Maps REST Imagery Service URL as the source to an image tag.

Implement using JavaScript

Implementing the Bing Maps REST services in JavaScript is fairly easy. All we have to do is create a request URL, and use the **WinJS.xhr** function to download the response. Once downloaded we can take the response text and parse it into a JSON object. The following is an example of how to do this.

```
var geocodeUri = "http://dev.virtualearth.net/REST/v1/Locations?q=" +
  encodeURIComponent(args.queryText) + "&key=" + credentials;

//Get response from Bing Maps REST services
WinJS.xhr({ url: geocodeUri }).then(function (e) {
  //Parse the text response into JSON
  var r = JSON.parse(e.responseText);

  //Do something with the response data
});
```

Implement using .NET

Implementing the Bing Maps REST services in .NET require a bit more work as we need to convert the responses into a usable .NET object. To do this we first need to create a set of classes that can be used to deserialize the JSON responses into a usable .NET object. These type of classes, which are used to deserialize the response of a service, are commonly known as data contracts. To create the data contracts first add a class file to your project and delete all the code inside of it. Then go to the MSDN documentation (<http://bit.ly/12Dr8mm>) and copy the JSON data contract for C# or Visual Basic and paste it into this empty class file. I've pointed to documentation for the data contracts rather than simply adding them to the appendix of this book as they get updated as new features are added to the Bing Maps REST services. Note that updates to the Bing Maps REST services do not break apps using old data contracts so there is no need to worry about updating your app every time a new feature is added to these services, only when you want to make use of new features. Now that have these data contracts in your app you can now deserialize the responses from the Bing Maps REST services.

Making HTTP Get Requests

To make an HTTP Get request against the Bing Maps REST service we simply need to create properly formatted URL and download the content it points to. To understand the response we need to deserialize it using the JSON data contracts. In order to do this we can use the **DataContractJsonSerializer** class by referencing the following libraries into our project (via the **using** or **Imports** keywords).

- **System.Runtime.Serialization**
- **System.ServiceModel.Web**

The following is a generic method that can be used to make a GET requests with the Bing Maps REST Services and deserialize the response.

C#

```
private async Task<Response> GetResponse(Uri uri)
{
    System.Net.Http.HttpClient client = new System.Net.Http.HttpClient();
    var response = await client.GetAsync(uri);

    using (var stream = await response.Content.ReadAsStreamAsync())
    {
        DataContractJsonSerializer ser = new DataContractJsonSerializer(typeof(Response));
        return ser.ReadObject(stream) as Response;
    }
}
```

Visual Basic

```
Private Async Function GetResponse(uri As Uri) As Task(Of Response)
    Dim client As New System.Net.Http.HttpClient()
    Dim response = Await client.GetAsync(uri)

    Using stream = Await response.Content.ReadAsStreamAsync()
        Dim ser As New DataContractJsonSerializer(GetType(Response))
        Return TryCast(ser.ReadObject(stream), Response)
    End Using
End Function
```

You can then make requests against the Bing Maps REST service using this newly created **GetResponse** method like this:

C#

```
Uri geocodeUri = new Uri(
    string.Format("http://dev.virtualsearch.net/REST/v1/Locations?q={0}&key={1}",
        Uri.EscapeUriString(QueryText), Credentials));

//Get response from Bing Maps REST services
Response r = await GetResponse(geocodeUri);

//Do something with the response data
```

Visual Basic

```
Dim geocodeUri As Uri = New Uri(
    String.Format("http://dev.virtualsearch.net/REST/v1/Locations?q={0}&key={1}",
        Uri.EscapeUriString(QueryText), Credentials))

'Get response from Bing Maps REST services
Dim r As Response = Await GetResponse(geocodeUri)

'Do something with the response data
```

Geocoding with the Search Charm

In this code sample we are going to create a simple mapping application that allows the user to search for locations using the Search charm. To do this there are three main tasks that have to be done:

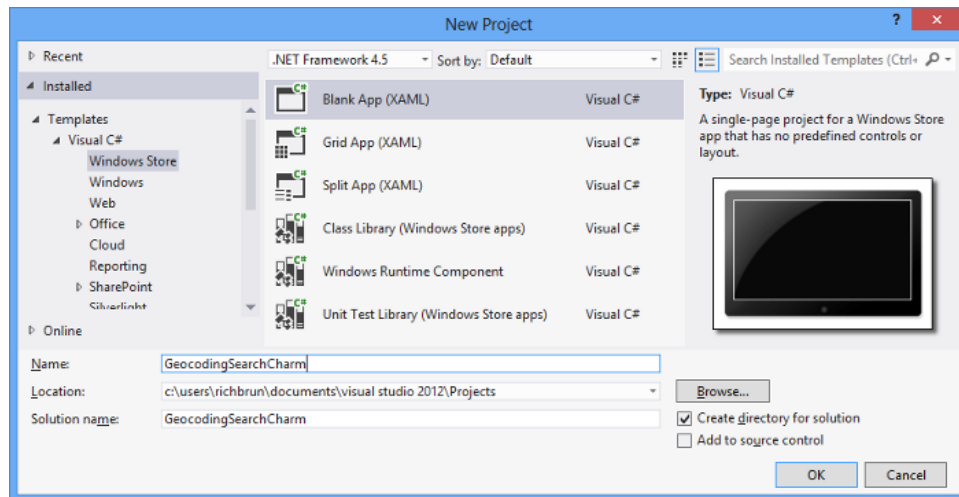
- 1) Create a basic mapping app
- 2) Integrate with the search charm
- 3) Add geocoding logic to the search charm

By making use of the search charm users can search within your app from anywhere in their system, even if it is not the main application currently open.

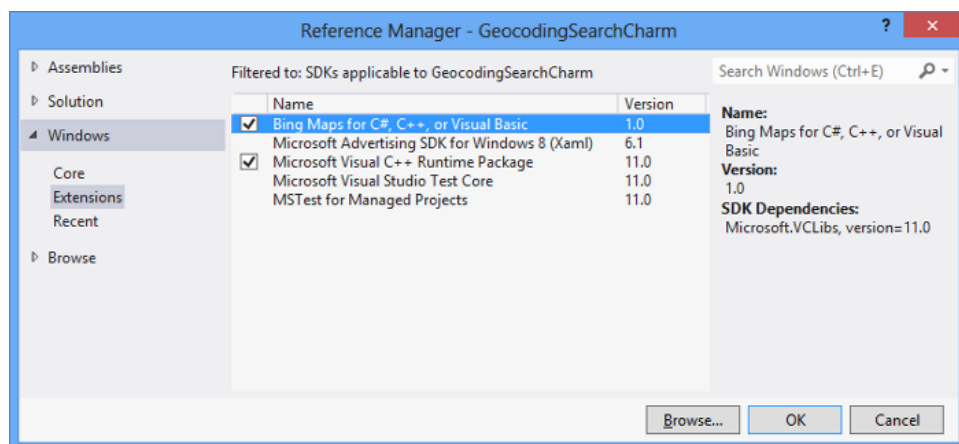
Creating a basic mapping app

Before diving into adding logic for tying into the search charm let's start off with creating a basic mapping application which will be used to display the results on.

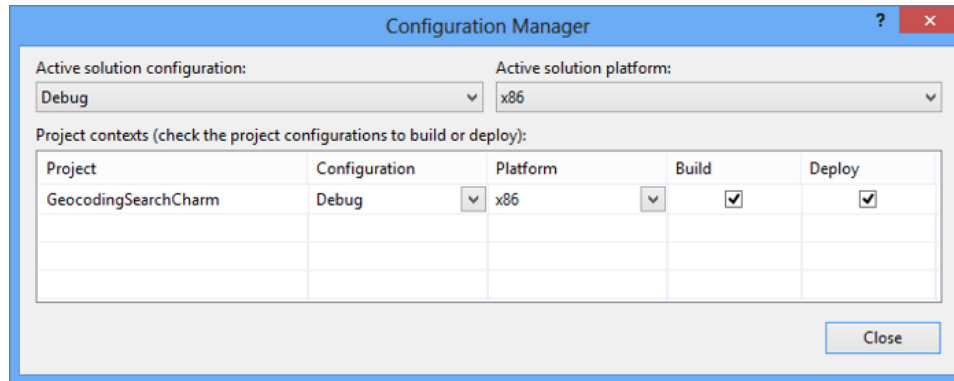
To get started open up Visual Studios and create a new project in your preferred language: JavaScript, C# or Visual Basic. Select the **Blank App** template and call the application **GeocodingSearchCharm** and press OK.



Add a reference to the Bing Maps SDK. To do this, right click on the **References** folder and press **Add Reference**. Select **Windows** -> **Extensions** select **Bing Maps for C#, C++ and Visual Basic** or **Bing Maps for JavaScript**. If you do not see this option ensure that you have installed the Bing Maps SDK for Windows Store apps. While you are here, if using C# or Visual Basic, add a reference to the **Microsoft Visual C++ Runtime Package** as this is required by the Bing Maps SDK.



If you are using C# or Visual Basic you may notice that there is a little yellow indicator on the references that you just added. The reason for this is that in the C++ runtime package you have to set the **Active solution platform** in Visual Studio to one of the following options; **ARM**, **x86** or **x64**. To do this, right click on the Solution folder and select **Properties**. Then go to **Configuration Properties -> Configuration**. Find your project and under the **Platform** column set the target platform. For this blog post I'm going to select x86. Press Ok and the yellow indicator should disappear from our references.



If you are using JavaScript right click on the **js** folder and select **Add -> New Item**. Create a new JavaScript file called **GeocodeSearch.js**. We will put all our JavaScript for this application in there to keep things clean. Now open up the **default.html** file and update the HTML to the following:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>GeocodingSearchCharm</title>

  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.1.0/css/ui-dark.css" rel="stylesheet" />
  <script src="//Microsoft.WinJS.1.0/js/base.js"></script>
  <script src="//Microsoft.WinJS.1.0/js/ui.js"></script>

  <!-- BingMapsShareCharm references -->
  <link href="/css/default.css" rel="stylesheet" />
  <script src="/js/default.js"></script>

  <!-- Bing Map Control references -->
  <script type="text/javascript" src="ms-appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>

  <!-- Our Bing Maps JavaScript Code -->
  <script src="/js/GeocodeSearch.js"></script>
</head>
<body>
  <div id="MyMap"></div>
</body>
</html>
```

Now open the **GeocodeSearch.js** file and add the following code to load the map and generate a session key for use later on.

```
(function () {
  var map, sessionKey;

  function initialize() {
    Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: GetMap });
  }

  function GetMap() {
    map = new Microsoft.Maps.Map(document.getElementById("MyMap"), {
      credentials: "YOUR_BING_MAPS_KEY"
    });
  }
});
```



```

        map.getCredentials(function (c) {
            sessionKey = c;
        });
    }

    document.addEventListener("DOMContentLoaded", initialize, false);
})();

```

If you are using C# or Visual Basic open the **MainPage.xaml** file and add update the XAML to the following:

```

<Page
    x:Class="GeocodingSearchCharm.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:GeocodingSearchCharm"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:m="using:Bing.Maps">

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <m:Map Name="MyMap" Credentials="YOUR_BING_MAPS_KEY"/>
    </Grid>
</Page>

```

At this point if you run the application you will end up with a Bing map that takes up the whole screen and looks like this.



Integrating with the Search Charm

Next we can add the functionality to power the search charm. If you are using JavaScript you will first need to add a Search declaration to the project. To do this open the app manifest add go to **Declarations** tab. From the available

declarations dropdown select **Search** and press add. This will register your app as a search provider. This will allow end users to search against your app from anywhere in the system.

Now open the **default.js** file. In here we will want to get a reference to the search pane and attach an event handler for when the user makes a search against the app. We will also want to remove this event handler when the user navigates away from the app. Open the **default.js** file and update it so that it looks like this:

```
(function () {
    var app = WinJS.Application;
    var map, sessionKey, searchPane;

    app.onready = function (args) {
        //Get the Search Pane object associated with current window
        searchPane = Windows.ApplicationModel.Search.SearchPane.getForCurrentView();

        //Add watermark text to the search textbox
        searchPane.placeholderText = "Search for an address, city or place.";
        searchPane.showOnKeyboardInput = true;

        //Register the share handler event
        searchPane.addEventListener("querysubmitted", SearchHandler);
    };

    app.onunload = function (args) {
        //Unregister the share handler event
        searchPane.removeEventListener("querysubmitted", SearchHandler);
    };

    function initialize() {
        Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: GetMap });
    }

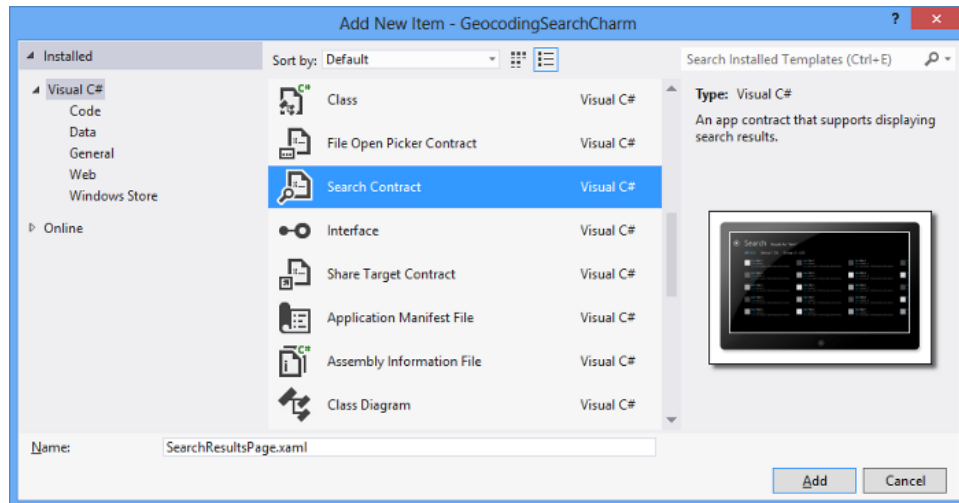
    function GetMap() {
        map = new Microsoft.Maps.Map(document.getElementById("MyMap"), {
            credentials: "YOUR_BING_MAPS_KEY"
        });

        map.getCredentials(function (c) {
            sessionKey = c;
        });
    }

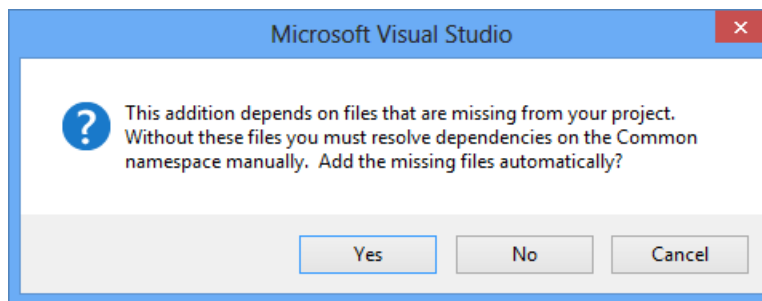
    function SearchHandler(args) {
        //TODO: Add logic for geocoding the query text.
    }

    document.addEventListener("DOMContentLoaded", initialize, false);
})();
```

If using C# or Visual Basic add a search contract to the project. To do this right click on the project and select **Add -> New Item**. Locate the search contract item and called **searchResults.html** or **SearchResultsPage1.xaml**.



An alert will be displayed that asks if you would like to add some missing files to your project. Press Yes.



At this point you will notice a number of changes have been made to the project.

- In the app manifest the Search contract is declared under the **Declarations** tab.
- A basic search results page is created for your app. This results page adheres to the UX guidelines for result pages in the Microsoft guidelines and checklist for search.
- If using C# or Visual Basic an **OnSearchActivated** event handler is added to the **App.xaml.cs** file. This allows your application to respond to search queries, even when your app isn't the main app on the screen. And a number of helper files are added under the Common folder.

You actually don't need the **SearchCharmResults1.xaml** file, only all the additional changes that adding it to the project made. So delete this file from the project. Open the **App.xaml.cs** file and locate the **OnSearchActivated** event handler. Near the end of this event handler find the following line of code:

```
frame.Navigate(typeof(SearchCharmResults1), args.QueryText);
```

Replace the **SearchCharmResults1** value with **MainPage**. This will cause the main application to start when performing a search against it, even if the app isn't loaded.

Open the **MainPage.xaml.cs** file. In here you will want to get a reference to the search charm panel and attach a **QuerySubmitted** event when the user navigates to the app. You will also want to remove this event handler when the user navigates away from the app.

C#

```
using Bing.Maps;
using BingMapsRESTService.Common.JSON;
```

```

using System;
using System.Runtime.Serialization.Json;
using System.Threading.Tasks;
using Windows.ApplicationModel.Search;
using Windows.UI.Popups;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;

namespace GeocodingSearchCharm
{
    public sealed partial class MainPage : Page
    {
        private SearchPane searchPane;
        private string sessionKey;

        public MainPage()
        {
            this.InitializeComponent();

            //Get session key from map after it is loaded
            MyMap.Loaded += async (s, a) =>
            {
                try
                {
                    sessionKey = await MyMap.GetSessionIdAsync();
                }
                catch { }
            };

            //Get a reference to the Search charm
            searchPane = SearchPane.GetForCurrentView();

            //Add watermark text to the search textbox
            searchPane.PlaceholderText = "Search for an address, city or place.";
            searchPane.ShowOnKeyboardInput = true;
        }

        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            base.OnNavigatedTo(e);

            searchPane.QuerySubmitted += searchPane_QuerySubmitted;
        }

        protected override void OnNavigatedFrom(NavigationEventArgs e)
        {
            base.OnNavigatedFrom(e);

            searchPane.QuerySubmitted -= searchPane_QuerySubmitted;
        }

        private void searchPane_QuerySubmitted(SearchPane sender,
        SearchPaneQuerySubmittedEventArgs args)
        {
            //TODO: Add logic for geocoding the query text.
        }
    }
}

```

```

Imports Bing.Maps
Imports GeocodingSearchCharm.BingMapsRESTService.Common.JSON
Imports System
Imports System.Runtime.Serialization.Json
Imports System.Threading.Tasks
Imports Windows.ApplicationModel.Search
Imports Windows.UI.Popups
Imports Windows.UI.Xaml.Controls
Imports Windows.UI.Xaml.Navigation

Public NotInheritable Class MainPage
    Inherits Page

    Private searchPane As SearchPane
    Private sessionKey As String

    Public Sub New()
        Me.InitializeComponent()

        'Get session key from map after it is loaded
        AddHandler MyMap.Loaded, Async Sub(s, a)
            Try
                sessionKey = Await MyMap.GetSessionIdAsync()
            Catch
            End Try
        End Sub

        'Get a reference to the Search charm
        searchPane = searchPane.GetForCurrentView()

        'Add watermark text to the search textbox
        searchPane.PlaceholderText = "Search for an address, city or place."
        searchPane.ShowOnKeyboardInput = True
    End Sub

    Protected Overrides Sub OnNavigatedTo(e As Navigation.NavigationEventArgs)
        MyBase.OnNavigatedTo(e)

        AddHandler searchPane.QuerySubmitted, AddressOf searchPane_QuerySubmitted
    End Sub

    Protected Overrides Sub OnNavigatedFrom(e As NavigationEventArgs)
        MyBase.OnNavigatedFrom(e)

        RemoveHandler searchPane.QuerySubmitted, AddressOf searchPane_QuerySubmitted
    End Sub

    Private Async Sub searchPane_QuerySubmitted(sender As SearchPane, args As
SearchPaneQuerySubmittedEventArgs) As Task
        'TODO: Add logic for geocoding the query text.
    End Sub
End Class

```

Adding the Geocoding logic

At this point we have two of the three main tasks completed. The next step is to integrate the geocoding logic by using the Bing Maps REST services. If using JavaScript the only step remaining is to add the logic to the **SearchHandler** method as shown below.

```

function SearchHandler(args) {
    //Remove any existing search result items from the map.

```

```

map.entities.clear();

//Logic for geocoding the query text.
if (args.queryText && args.queryText != '') {
    var geocodeUri = "http://dev.virtualearth.net/REST/v1/Locations?q=" +
        encodeURIComponent(args.queryText) + "&key=" + sessionKey;

    //Get response from Bing Maps REST services
    WinJS.xhr({ url: geocodeUri }).then(function (e) {
        //Parse the text response into JSON
        var r = JSON.parse(e.responseText);

        if (r != null &&
            r.resourceSets != null &&
            r.resourceSets.length > 0 &&
            r.resourceSets[0].resources != null &&
            r.resourceSets[0].resources.length > 0) {
            var locations = [];

            var list = new WinJS.Binding.List(r.resourceSets[0].resources);
            list.forEach(function(val, index){
                //Get the location of each result
                var location = new Microsoft.Maps.Location(val.point.coordinates[0],
                    val.point.coordinates[1]);

                //Create a pushpin for each location
                var pin = new Microsoft.Maps.Pushpin(location,
                    {
                        text: (index + 1) + '
                    });

                pin.Tag = val.name;

                //Add a click event that will display the name of the location
                Microsoft.Maps.Events.addHandler(pin, "click", function (e) {
                    if (e.target && e.target.Tag) {
                        new Windows.UI.Popups.MessageDialog(e.target.Tag).showAsync();
                    }
                });

                //Add the pushpin to the map
                map.entities.push(pin);

                //Add the coordinates of the location to a location collection
                locations.push(location);
            });

            //Set the map view based on the location collection
            map.setView({ bounds: Microsoft.Maps.LocationRect.fromLocations(locations)
        });
    }
    else {
        new Windows.UI.Popups.MessageDialog("No results found.").showAsync();
    }
});
}
}

```

If using C# or Visual Basic you will need to add the JSON data contracts for the Bing Maps REST services to your project. To do this right click on your project and select **Add -> New Item**. Create a class called

BingMapsRESTServices.cs. Open the file and delete the contents. Copy the appropriate JSON data contracts for the Bing Maps REST services from the MSDN documentation (<http://bit.ly/12Dr8mm>). Paste the JSON data contracts into the **BingMapsRESTServices.cs** file. In the **MainPage.xaml.cs** file add the following helper method for handling requests to the Bing Maps REST services.

C#

```
private async Task<Response> GetResponse(Uri uri)
{
    System.Net.Http.HttpClient client = new System.Net.Http.HttpClient();
    var response = await client.GetAsync(uri);

    using (var stream = await response.Content.ReadAsStreamAsync())
    {
        DataContractJsonSerializer ser = new
        DataContractJsonSerializer(typeof(Response));
        return ser.ReadObject(stream) as Response;
    }
}
```

Visual Basic

```
Private Async Function GetResponse(uri As Uri) As Task(Of Response)
    Dim client As New System.Net.Http.HttpClient()
    Dim response = Await client.GetAsync(uri)

    Using stream = Await response.Content.ReadAsStreamAsync()
        Dim ser As New DataContractJsonSerializer(GetType(Response))
        Return TryCast(ser.ReadObject(stream), Response)
    End Using
End Function
```

Next the logic for creating the geocoding request can be added to the **searchPane_QuerySubmitted** event handler. Once a request has been made you can loop through the results and add them as pushpins on the map. Update the **searchPane_QuerySubmitted** event handler such that it looks like this:

C#

```
private async void searchPane_QuerySubmitted(SearchPane sender,
SearchPaneQuerySubmittedEventArgs args)
{
    try
    {
        //Remove any existing search result items from the map.
        MyMap.Children.Clear();

        //Logic for geocoding the query text.
        if (!string.IsNullOrEmpty(args.QueryText))
        {
            Uri geocodeUri = new Uri(
string.Format("http://dev.virtualearth.net/REST/v1/Locations?q={0}&c={1}&key={2}",
Uri.EscapeUriString(args.QueryText), args.Language, sessionKey));

            //Get response from Bing Maps REST services
            Response r = await GetResponse(geocodeUri);

            if (r != null &&
```



```

        r.ResourceSets != null &&
        r.ResourceSets.Length > 0 &&
        r.ResourceSets[0].Resources != null &&
        r.ResourceSets[0].Resources.Length > 0)
    {
        LocationCollection locations = new LocationCollection();

        int i = 1;

        foreach (BingMapsRESTService.Common.JSON.Location l
                 in r.ResourceSets[0].Resources)
        {
            //Get the location of each result
            Bing.Maps.Location location =
                new Bing.Maps.Location(l.Point.Coordinates[0],
l.Point.Coordinates[1]);

            //Create a pushpin each location
            Pushpin pin = new Pushpin()
            {
                Tag = l.Name,
                Text = i.ToString()
            };

            i++;

            //Add a tapped event that will display the name of the location
            pin.Tapped += async (s, a) =>
            {
                try
                {
                    var p = s as Pushpin;
                    await new ProgressDialog(p.Tag as string).ShowAsync();
                }
                catch { }
            };

            //Set the location of the pushpin
            MapLayer.SetPosition(pin, location);

            //Add the pushpin to the map
            MyMap.Children.Add(pin);

            //Add the coordinates of the location to a location collection
            locations.Add(location);
        }

        //Set the map view based on the location collection
        MyMap.SetView(new LocationRect(locations));
    }
    else
    {
        await new ProgressDialog("No results found.").ShowAsync();
    }
}
}
catch { }
}

```

```

Private Async Sub searchPane_QuerySubmitted(sender As SearchPane, args As
SearchPaneQuerySubmittedEventArgs)
    Try
        'Remove any existing search result items from the map.
        MyMap.Children.Clear()

        'Logic for geocoding the query text.
        If Not String.IsNullOrEmpty(args.QueryText) Then

            Dim geocodeUri As Uri = New Uri(
String.Format("http://dev.virtualsearth.net/REST/v1/Locations?q={0}&c={1}&key={2}",
Uri.EscapeUriString(args.QueryText), args.Language, sessionKey))

            'Get response from Bing Maps REST services
            Dim r As Response = Await GetResponse(geocodeUri)

            If r IsNot Nothing AndAlso r.ResourceSets IsNot Nothing AndAlso
r.ResourceSets.Length > 0 AndAlso r.ResourceSets(0).Resources IsNot Nothing AndAlso
r.ResourceSets(0).Resources.Length > 0 Then
                Dim locations As New LocationCollection()

                Dim i As Integer = 1

                For Each l As BingMapsRESTService.Common.JSON.Location In
r.ResourceSets(0).Resources
                    'Get the location of each result
                    Dim location As New Bing.Maps.Location(l.Point.Coordinates(0),
l.Point.Coordinates(1))

                    'Create a pushpin each location
                    Dim pin As New Pushpin()
                    pin.Tag = l.Name
                    pin.Text = i.ToString()

                    i += 1

                    'Add a tapped event that will display the name of the location
                    AddHandler pin.Tapped, Async Sub(s, a)
                        Try
                            Dim p = TryCast(s, Pushpin)
                            Await New MatDialog(TryCast(p.Tag,
String)).ShowAsync()
                        Catch
                            End Try
                        End Sub

                    'Set the location of the pushpin
                    MapLayer.SetPosition(pin, location)

                    'Add the pushpin to the map
                    MyMap.Children.Add(pin)

                    'Add the coordinates of the location to a location collection
                    locations.Add(location)
                Next

                'Set the map view based on the location collection
                MyMap.SetView(New LocationRect(locations))
            Else
                Await New MatDialog("No Results found.").ShowAsync()
            End If
        End Try
    End Sub

```

```

        End If
    End If
Catch
End Try
End Sub

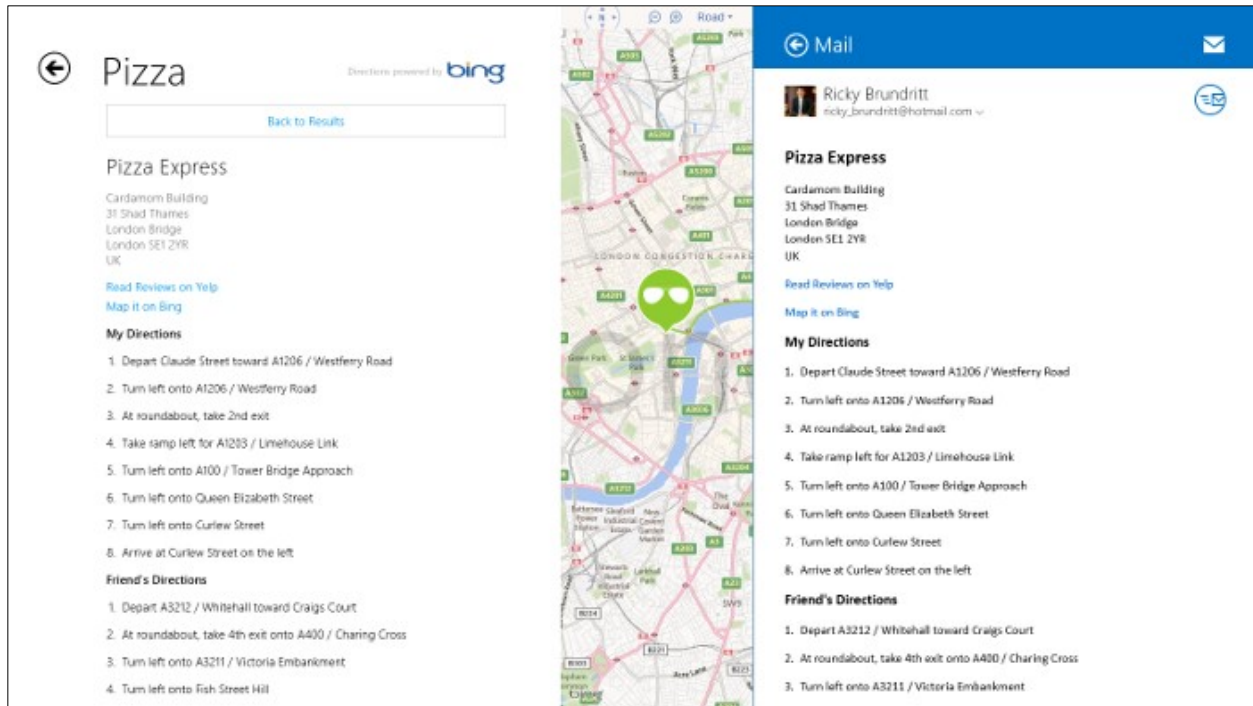
```

At this point the application is complete. The final step is to run and test the application. Press the Debug button to launch the application. Open up the search charm from the right side panel and perform a search. Here is a screenshot of a search from London.



Real World Example: Meet me in the Middle

Meet me in the Middle is a simple app to help you determine where to meet your friends for fun, food, and outdoor activities. Simply launch the app, select what you want to do, and the app uses the power of Yelp and Bing Maps to find a place for you to meet that is approximately equidistance from you and your friend. Bing Maps is being used in this app to provide the mapping and directions functionalities. The routing service is being used to calculate the driving distance and mid-point between you and your friend. This mid-point is then used to do a nearby search against Yelp's point of interest data. Once a location is selected, directions are displayed to that location and can be shared. When sharing directions the instructions are added to an email as HTML along with a link that opens the location on the Bing Maps website, thus allowing you to share an interactive map with your friend. You can find this app in the Windows Store here: <http://bit.ly/1fyBvNw>



Chapter Summary

In this chapter you were introduced to the Bing Maps REST Services along with searching and sharing content within a Windows Store app.

- The Bing Maps REST Services has a number of different APIs that include the following main functionalities; Geocoding, Reverse Geocoding, Routing, Static Imagery, Traffic and Elevation data.
- The Bing Maps SOAP services are a legacy service and is not generally recommended for new applications as it is slower and less accurate than the Bing Maps REST services.
- You can find full documentation for the Bing Maps REST services here: <http://bit.ly/10JoNVW>
- The REST services can return results as JSON (default) or XML. JSON is normally 20%-30% smaller than XML which means it will download faster.
- When testing the REST services you can simply place a URL in a web browser. You may find it useful to return the results as XML (&output=xml) for better readability.
- Use the culture parameter when geocoding or routing between addresses that are not in English. This will help the service find the best results and will also return directions in the specified language.
- Encode addresses and query parameters that are added to a request URL. This will ensure that special characters are handled properly. Be sure not to encode coordinates.
- When reverse geocoding coordinates, round the latitude and longitude values to 5 or 6 decimal places. 6 decimal places has approximately 10cm accuracy, any more decimal places is just a waste and increases the chances confusing the service into thinking that this is a geocoding request.
- You can share links to the Bing Maps consumer site using URL's similar to the Bing Maps REST service. You can find full documentation on this here: <http://bit.ly/1aHCrng>
- You can create a Bing Maps session key from the map. Session keys are a great way to reduce the number of billable transactions your application generates.
- You can easily use the REST services in .NET by using the **DataContractJsonSerializer** with the Bing Maps REST service data contracts which you can find here: <http://bit.ly/12Dr8mm>

- You can easily share images of maps by using the Imagery service and the Share charm in the Windows Store app.
- You can allow users to easily search within your app, even when it is not main app in view, by using the Search Charm in the Windows Store app.

Chapter 6: Bing Spatial Data Services

This Bing Spatial Data Services exposes a number of API's for performing a number of different tasks. Here is a list of the API's available in this service.

API	Description
Geocode DataFlow API	This API provides a set of tools to perform batch geocoding and reverse geocoding of large data sets.
GeoData API	This API provides access to polygons that describe the boundaries of geographical regions including; counties, states, provinces, zip codes and postal codes.
Data Source Management API	This API provides a set of tools to create, edit, upload and store custom spatial data on the Bing Maps servers.
Query API	This API exposes the data stored on the Bing Maps servers as a REST service and provides a number of useful spatial queries such as find nearby and find along a route.

In addition to these API's the Bing Spatial Data Services also provides access to NAVTEQ point of interest data and Real time traffic incident information. This data is stored in the Data Source Management API and exposed through the Query API.

The Bing Maps Portal

The Bing Maps Portal (<http://bingmapsportal.com>) is the primary place where Bing Maps developers can manage their account. Those developers who have access to an account that is licensed as an Enterprise Bing Maps account will find they have access to a set of tools for uploading data sets to be batch geocoded or stored in through the Data Source Management API. This premium service allows Enterprise account users with the ability perform these tasks without having to write any code against the Geocode DataFlow or Data Source Management API's. This is one of the benefits of having an Enterprise accounts. If you are working with an Enterprise account you can find documentation on how to use these tools in the Bing Maps portal here: <http://bit.ly/19eVWxT>

API Usage Limits

When accessing the Geocode Data Flow or the Data Source Management tools either through the Bing Map Portal or their API's a job is created to perform the requested action. Every account has a limits on how much this API can be used, however these limits vary depending on the type of Bing Maps account and key being used.

The following limits are in place for Basic or Trial keys that are used from the same Bing Maps Account:

- A maximum of 2 jobs can be in process at the same time.
- No more than 5 jobs can be run within a 24 hour period.
- A maximum of 5 data sources can be created per Bing Maps Account.
- Data that is geocoded or uploaded to a data source must use UTF-8 encoding, and can have a maximum of 50 entities (rows of data). Compressed data files are accepted.
- Geocode results can be downloaded for up to 14 calendar days after a geocode job completes.

The following limits are in place for Enterprise keys that are used from the same Bing Maps Account. Jobs that use Basic or Trial keys that belong to an Enterprise account have the limits described above and are also included in the following limits.

- A maximum of 10 jobs can be in process at the same time. This limit also includes jobs created using a Basic and Trial keys.
- No more than 50 jobs can be run in a 24 hour period. This limit also includes jobs created using a Basic and Trial keys.
- A maximum of 25 data sources can be created per Bing Maps Account.
- Data that is geocoded or uploaded to a data source must use UTF-8 encoding, and can have up to 300 MB of uncompressed data and a maximum of 200,000 entities (rows of data). Compressed data files are accepted, but the uncompressed limit applies.
- Geocode results can be downloaded for up to 14 calendar days after a geocode job completes.

These limits may change over time. Check the MSDN documentation (<http://bit.ly/18jPdCG>) and the Bing Maps Terms of Use (<http://www.microsoft.com/maps/product/terms.html>) for the current API limits.

Bing Spatial Data Services Excel Add-in

The Bing Spatial Data Services Excel add-in is an open source project available on CodePlex (<http://bsdsexceladdin.codeplex.com>) that allows you to use the batch geocode, reverse geocode and manage your data sources directly from Excel. This add-in is supports Excel 2010 and above and solves a lot of issues that are typically encountered when trying to manage Bing Spatial data sources in Excel. When installed this will add a new tab to Excel which will have a ribbon that includes the following features:

- Import CSV, Pipe, Tab and XML formatted data sources
- Export Excel sheet as Bing spatial Data Source in CSV, Pipe, tab and XML format
- Batch geocode and reverse geocode directly in Excel
- Upload Data Source directly from Excel
- Store data source information; name, entity type, master/query keys in workbook to save time when uploading
- Data Source validation (verify that it is properly formatted)
- Basic data source template
- Manage data sources by loading all data sources for an account Delete
 - Change public setting on data source
 - Get Query URL
 - Download Data Source (downloads it into an Excel Workbook)
- Manage Dataflow Jobs and view details of each job.

The download includes full instructions on how to install and use this add-in. Note that the full source code is available for the add-in if you wish to see how it uses the Bing Spatial Data Services API's.

If you do not have an Enterprise Bing Maps account or you manage your data in Excel then this is the easiest way to make use of the Geocode DataFlow and Data Source Management API's without having to write any code. Note that this add-in currently only supports point based data and not the new Geometry data type available in data sources.

Geocode DataFlow API

The Geocode DataFlow API provides a set of tools to perform batch geocoding and reverse geocoding of large data sets. The batch process allows you to submit multiple locations to be geocoded or reverse geocoded in a single request. This service

is designed to be a batch process which means that when the job is created it is added to a queue and is scheduled to be processed. The length of time required to complete the batch process depends on the size of the file sent to be processed and the number of other jobs in the queue. Some jobs may process within minutes while others can take up to 24 hours. Combine this with the daily limits on the number of jobs that can be created per account and it should be easy to understand that this is not designed to be an on-demand service. This service is often used by developers to geocode all their locations once when they are initially creating their data set for their application. This service is also often used to develop a tool to batch geocode addresses that are stored in tables in SQL server. It's unlikely you will find yourself using this API inside of a Windows Store App but it's good to know it's available. You can find full documentation this API here: <http://bit.ly/1eVnM75>

Data Source Management API

The Data Source Management API provides tools for create, edit, upload and store custom spatial data sources on the Bing Maps Servers. Not only can this service be used to store point based information but it also supports complex geometries as well such as polygons. This API is typically used in backend tools and is not something you would likely use directly from a Windows Store App. The Bing Maps portal and the Bing Spatial Data Services Excel Add-in wraps this API to save you having to develop against this API, however, if you have a backend system that maintains your data it might be useful to implement this API to automatically upload updates. You can find full documentation on this API here: <http://bit.ly/1cljBB7>

When uploading a data source to Bing Maps you will be required to specify a master key. A master key is a Bing Maps key which has the right to edit, download and delete a data source. In addition to this you can optionally specify a query key, which is a Bing Maps key that can only read a data source. It is a best practice to define a query key for any data source which is going to be used in a public facing application. Also, if you use this same Bing Maps key to load the map you can use the session key generated from the map to access this data source. This will ensure that all requests using the Query API are tracked as part of the users session and thus marked as non-billable.

Tip: In addition to being able to make your data available to your application it is also possible to make data sources public. This provides anyone with read-only access to the data source using their own key. This is a great way to share your location data with others so that they help promote your locations.

Data Source Schema

Regardless of how you upload your data to the Bing Spatial Data Services it will need to conform to a specific data schema. The data schema defines the name and attributes of the data type to use for a data source. A data source can be created using XML, or a comma, tab or pipe (|) delimited file. In this book we will focus on the delimited file format as this is easy to create in Excel.

Tip: The XML data source schema is actually the XML representation of a .NET **DataSet** object from the **System.Data** namespace (not available in Windows Store Apps). The first **DataTable** in the data set contains the data source. The **DataSetName** property of the **DataSet** will contain the data source name. The **TableName** property of the **DataTable** will contain the entity type name. This can be used to programmatically create and parse data sources in standard .NET applications.

When creating a data source you must define a data source name and entity type name. These will be required to access your data from the Query API. The data source and entity type names can have up to 50 characters and can contain alphanumeric characters. An easy way to understand the difference and purpose of the data source and entity type names, think of the data source name as a "group" and the entity type name as the "type of group". If your data source contains a list of Starbucks coffee shops you might set the data source name to Starbucks and the entity type name to CoffeeShops. When creating a delimited data source file the first line must define the data schema version and the entity type name using comma's to separate the values. For example:

Bing Spatial Data Services, 1.0, CoffeeShops

Note that currently there is only one data schema version available, 1.0.

Defining Columns of Data

The name of each column and the data type it contains must be defined after the data schema version line. Each column name must meet the following requirements:

- The column name can have up to 50 characters (not including the data type name and primary key information).
- The column name must contain alphanumeric characters and underscores only.
- The first character of the property name must be a letter or an underscore.
- The column name cannot start with a two underscores, this is reserved for internal names.
- The string "Entity" is reserved and cannot be used as a column name.
- Column names are case-insensitive. So don't use the same name twice.
- Each column should have the data type specified in brackets in the same cell. For example:

Longitude(Edm.Double)

A data source can have up to a maximum of 350 properties defined in the schema, not including a Latitude and Longitude column. Each column requires a data type to be specified beside the name of the column. Here is a list of the different data types that can be used.

Data Type name	Restrictions
Edm.String	The maximum string length is 2560 characters.
Edm.Int64	
Edm.Boolean	
Edm.Double	
Edm.DateTime	
Edm.Geometry	The maximum number of points for any geography type is 2000 and defined using a well-known text.

A data source also requires that one column is used as a primary key that is unique for every entity in the data source. The data type of the primary key property must be a string and all primary key values must have a maximum length of 50 characters. The column being used should have a format like this: **ColumnName(Edm.String,primaryKey)**.

At a minimum a data source is required to have a primary key column and a location data column (s). The location data columns can consist two columns; Latitude, Longitude defined using the **Edm.Double** data type or a single geometry column defined using the **Edm.Geometry** data type. If these columns do not exist the data source will not pass validation and will not be uploaded to the Bing Spatial Data Services.

If you would like to be able to geocode address data that is in your data source the following columns names can be used to specify the different parts of an address using the **Edm.String** data type.

Column Name	Description
AddressLine	Street address
Locality	City or town name
PostalCode	Zip or postal code
AdminDistrict	State or province name
CountryRegion	Country name

Note that you do not need to specify all of these columns in your data source, only the ones you want to use for geocoding. You can find full documentation on creating a data source along with samples here: <http://bit.ly/IN3Fxb>

Geometry Data Type

The geometry data type can be used to define a geometry inside of a data source, or be used to perform intersection tests with the Query API. A geometry is defined using well-known text. Well-known text is a standard method for representing spatial objects as a string. The standard geometry classes that are supported are **Point**, **LineString**, **Polygon**, **MultiPoint**, **MultiLineString**, **MultiPolygon**, and **GeometryCollection**. We will take a closer look at well-known text and these geometry classes in Chapter 7.

When defining a geometry in a data source is must not have more the 2,000 coordinates. Also, if using CSV it should be enclosed with double quotes to ensure that any commas in the geometry do not get confused as a column separator.

Public Data Sources

Earlier in this chapter I briefly mentioned that you can make data sources publicly available. The Bing Maps team has used this framework to make several of their data sources publicly available for you to use. Here is a list of these data sources:

Data Source Name	Description
FourthCoffeeSample	This data source contains sample data which is useful when testing the APIs.
NAVTEQNA	This data source contains points of interest for North America.
NAVTEQEU	This data source contains points of interest for Europe.
TrafficIncidents	This data source contains traffic incident data in areas where Bing Maps has traffic incident data.

To access any data source you need a base query URL which can be used with the Query API. These URLs have the following format:

`http://spatial.virtualearth.net/REST/v1/data/accessID/dataSourceName/entityTypeName`

The **accessID** is an alphanumeric value used to uniquely identify your data source. This allows other developers to have data sources that have the same data source and entity type name as yours without any conflicts occurring. The following is a list of values that you can use to create the query URL for these data sources.

Data Source Name	accessID/dataSourceName/entityTypeName
FourthCoffeeSample	20181f26d9e94c81acdf9496133d4f23/FourthCoffeeSample/FourthCoffeeShops
NAVTEQNA	f22876ec257b474b82fe2ffcb8393150/NavteqNA/NavteqPOIs
NAVTEQEU	c2ae584bbccc4916a0acf75d1e6947b4/NavteqEU/NavteqPOIs
TrafficIncidents	8F77935E46704C718E45F52D0D5550A6/TrafficIncidents/TrafficIncident

Note that these URLs support both http and https. All of these data sources have different schemas which you can find defined here: <http://bit.ly/IHk9q0>

Query API

The Query API exposes the data stored on the Bing Maps servers as a REST service. This service can be used to query locations using different queries such as find nearby, find by bounding box, find along a route, intersects with a geometry, or find by property. Every data source that is uploaded to the Bing Spatial Data Services is exposed using a

base query URL as we saw in the previous section. Using this base URL a spatial query can be added to it perform a spatial search against the data source. These query URLs have the following format:

`http://spatial.virtualearth.net/REST/v1/data/accessID/dataSourceName/entityTypeName?[queryOptions]&key=bingMapsKey`

Note that a Bing Maps key must be provided in the query to authenticate the request. If you trying to connect to a custom data source this will need to be either the master or query key that you used when uploading the data source.

When making a query there are a number of common query option parameters which can be used as defined below.

Query Option	Description																
\$filter	<p>Specifies a conditional expression to filter by property value. Filters cannot be applied to the latitude and longitude properties of a data source. The following logical operators can be used:</p> <table> <tr><td>Eq</td><td>Equal</td></tr> <tr><td>Ne</td><td>Not equal</td></tr> <tr><td>Gt</td><td>Greater than</td></tr> <tr><td>Ge</td><td>Greater than or equal</td></tr> <tr><td>Lt</td><td>Less than</td></tr> <tr><td>And</td><td>Logical And</td></tr> <tr><td>Or</td><td>Logical Or</td></tr> <tr><td>Not</td><td>Logical negation</td></tr> </table>	Eq	Equal	Ne	Not equal	Gt	Greater than	Ge	Greater than or equal	Lt	Less than	And	Logical And	Or	Logical Or	Not	Logical negation
Eq	Equal																
Ne	Not equal																
Gt	Greater than																
Ge	Greater than or equal																
Lt	Less than																
And	Logical And																
Or	Logical Or																
Not	Logical negation																
\$format	Specifies the format of the HTTP response. The supported formats for the Query API are JSON and Atom. The default format is Atom.																
jsonp	Name of a JSON callback function that is called when the response to the request is received. The JSON object provided in the response is passed to the callback function.																
jsono	The state object to pass to the JSON callback function. You can use a state object to match a response with a specific call. This value is provided as the second parameter to the callback function provided in the JSONP parameter.																
\$inlinecount	<p>Specifies whether or not to return a count of the results in the response. Possible values for this query option include:</p> <ul style="list-style-type: none"> • allpages • none [default] 																
\$orderby	<p>Specifies one or more properties to use to sort the results of a query. You can specify up to three properties. Results are sorted in ascending order.</p> <p>Each type of query, such as Query by ID or Query by Property, has a default sort order for query results. If the \$orderby option is not specified, query results are sorted using the default sort for that query type.</p> <p>The \$orderby option cannot be applied to the latitude and longitude properties of a data source.</p>																
\$select	<p>Specifies one of the following:</p> <ol style="list-style-type: none"> 1. The data source properties to return in the response. If the \$select query option is not specified or if it is set to "*" (\$select=*), all data source properties are returned. Example: \$select=Name,Phone 2. The intersection of the entity latitude or longitude or geographical area (defined by the entity geography) with the specified geography. If an entity has both latitude and longitude and geography properties in the schema, the intersection is computed with the geography property value. 																

\$skip	Specifies to not return a specified number of query results. For example, if this value is set to 50, then the first result that is returned is the 51st result. You can use the \$skip query option with the \$top query option to display a subset of the query results.
\$top	Specifies the maximum number of results to return in the query response. The default value is 25 and the maximum value is 250. You can return more than 250 results by making multiple queries and using the \$skip parameter to step through the results.

Full documentation on the Query API can be found here: <http://bit.ly/1clH315>

Find Nearby Query

A spatial filter can be used to perform nearby searches on data sources. This is often used to in queries where you want to find all the locations that are within a specific distance of a location. When performing nearby searches you will need to specify a latitude and longitude value for the center of the search and a radius in kilometers. The radius can be a distance between 0.16 and 1000 kilometers. This spatial filter can be added as a parameter to the query URL using the following format:

spatialFilter=nearby(latitude,longitude,radius)

Tip: The distance from the center point to each entity can be returned by specifying **__Distance** as one of the \$select query option values. This value will be in kilometers.

Find in Bounding Box Query

A spatial filter can be used to find all locations that are within a bounding box. This is useful if you want to find all locations that are in the current map view, by simply using the bounding box of the map to define the query. To define a bounding box spatial filter you need to specify the south latitude, west longitude, north latitude, and east longitude values. This spatial filter can be added as a parameter to the query URL using the following format:

spatialFilter=bbox(southLatitude,westLongitude,northLatitude,eastLongitude)

Find Along a Route Query

Using a near route spatial filter you can locate all locations in a data source that are within 1 mile or 1.6 kilometers of the route. When specifying a near route query a start and end location must be specified. These locations can be an address or a coordinate enclosed by single quotes. Coordinates should be formatted as 'latitude,longitude'. This spatial filter can be added as a parameter to the query URL using the following format:

spatialFilter=nearRoute(start,end)

If you wanted to perform a query along the route from coordinate (50, 0) to Paris the spatial filter would look like this:

spatialFilter=nearRoute('50,0','Paris')

When querying locations along a route additional query option parameters can be used to help define the type of route to calculate for the query. The following is a list of the more commonly used additional query options that are available for near route queries.

Parameter	Description
avoid	<p>A comma-separated list of values from the following list that limit the use of highways and toll roads in the route.</p> <ul style="list-style-type: none"> highways: Avoids the use of highways in the route. tolls: Avoids the use of toll roads in the route. minimizeHighways: Minimizes (tries to avoid) the use of highways in the route.

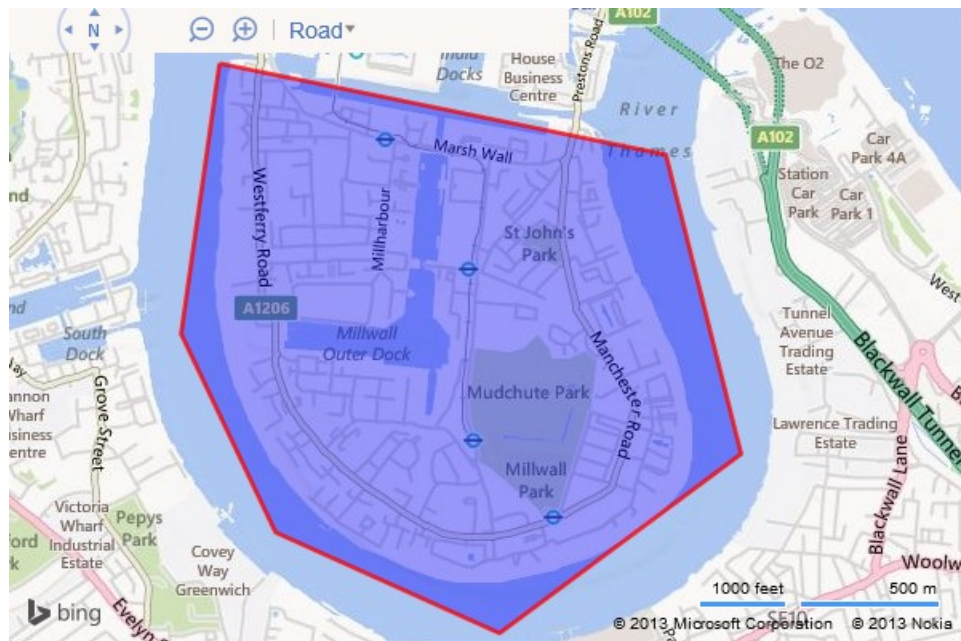
	<ul style="list-style-type: none"> • minimizeTolls: Minimizes (tries to avoid) the use of toll roads in the route.
optimize	Specifies how to optimize the route. <ul style="list-style-type: none"> • distance: The route is calculated to minimize the distance. Traffic information is not used. • time [default]: The route is calculated to minimize the time. Traffic information is not used. • timeWithTraffic: The route is calculated to minimize the time and uses current traffic information.
travelMode	Specifies method of transportation. <ul style="list-style-type: none"> • Driving [default] • Walking

Geometry Intersection Query

A spatial filter can be used to find all locations in a data source that intersect with a geometry. This may be useful if you want to find all locations within an arbitrary area. When performing this type of search you will need to specify a geometry that has no more than 200 coordinates as a well-known text value enclosed by single quotes. For performance there is a restriction that the geometry does not cover an area larger than 10 degrees of latitude and 10 degrees of longitude. This spatial filter can be added as a parameter to the query URL using the following format:

spatialFilter=intersects('geography well-known text')

Let's say you wanted to do a search for locations within the area drawn on the following map.



The spatial filter that you would need would look something like this:

```
spatialFilter=intersects('POLYGON((-0.028 51.503,-0.004 51.5,0 51.49,-0.013 51.484,-0.025 51.48737,-0.03 51.494,-0.028 51.503))')
```

For performance reasons this spatial filter has been disabled on the NAVTEQ point of interest data sources. If you try to use this spatial filter an error will be returned.

Implementing in JavaScript

Implementing the Query API in JavaScript is fairly easy. All we have to do is create a request URL, and use the **WinJS.xhr** function to download the response. Once downloaded we can take the response text and parse it into a JSON object. The following is an example of how to do this.

```
var queryUri = "[API query URL]";

//Get response from api
WinJS.xhr({ url: queryUri }).then(function (e) {
    //Parse the text response into JSON
    var r = JSON.parse(e.responseText);

    //Do something with the response data
});
```

Implement in .NET

Implementing the Query API in .NET requires a bit more work as we need to convert the responses into a usable .NET object. To do this we first need to create a set of classes that can be used to deserialize the JSON responses into a usable .NET object. These type of classes, which are used to deserialize the response of a service, are commonly known as data contracts. The JSON response from the Query API will vary based on the data source you are querying. The following is an example of a JSON response from the Query API.

```
{
  "d":{
    "__copyright":"","
    "results":[
      {
        "__metadata":{
          "uri":""
        },
        "Latitude":40.780329,
        "Longitude":-74.237863
      }
    ]
  }
}
```

You will have to create a set of data contracts to deserialize the response from your data source. The following is a very simple set of base data contracts which you can use as a starting point. Simply add additional properties to the **Result** class that align with the various columns and data types in your data source.

C#

```
using System.Runtime.Serialization;

namespace MyDataSourceSchema
{
    [DataContract]
    public class Response
    {
        [DataMember(Name = "d", EmitDefaultValue = false)]
        public ResultSet ResultSet { get; set; }
    }

    [DataContract]
    public class ResultSet
```



```

{
    [DataMember(Name = "__copyright", EmitDefaultValue = false)]
    public string Copyright { get; set; }

    [DataMember(Name = "results", EmitDefaultValue = false)]
    public Result[] Results { get; set; }
}

[DataContract]
public class Result
{
    [DataMember(Name = "Latitude", EmitDefaultValue = false)]
    public double Latitude { get; set; }

    [DataMember(Name = "Longitude", EmitDefaultValue = false)]
    public double Longitude { get; set; }
}
}

```

Visual Basic

```

Imports System.Runtime.Serialization

Namespace MyDataSourceSchema
    <DataContract> _
    Public Class Response
        <DataMember(Name:="d", EmitDefaultValue:=False)> _
        Public Property ResultSet As ResultSet
    End Class

    <DataContract> _
    Public Class ResultSet
        <DataMember(Name:="__copyright", EmitDefaultValue:=False)> _
        Public Property Copyright As String

        <DataMember(Name:="results", EmitDefaultValue:=False)> _
        Public Property Results As Result()
    End Class

    <DataContract> _
    Public Class Result
        <DataMember(Name:="Latitude", EmitDefaultValue:=False)> _
        Public Property Latitude As Double

        <DataMember(Name:="Longitude", EmitDefaultValue:=False)> _
        Public Property Longitude As Double
    End Class
End Namespace

```

The Query API is accessed by making a HTTP Get request against the service. To understand the response we need to deserialize it using the JSON data contracts. This can be done by using the **DataContractJsonSerializer** class and referencing in the following libraries into your project (via the **using** or **Imports** keywords).

- **System.Runtime.Serialization**
- **System.ServiceModel.Web**

The following is a modified version of the **GetResponse** method from Chapter 5 that can be used to make HTTP Get requests and deserialize the JSON response. This modified version makes use of generics which allows us to specify

the class type to serialize the response to. This is a lot more efficient than creating a separate method to deserialize each response type from all the services.

C#

```
private async Task<T> GetResponse<T>(Uri uri)
{
    System.Net.Http.HttpClient client = new System.Net.Http.HttpClient();
    var response = await client.GetAsync(uri);

    using (var stream = await response.Content.ReadAsStreamAsync())
    {
        DataContractJsonSerializer ser = new DataContractJsonSerializer(typeof(T));
        return (T)ser.ReadObject(stream);
    }
}
```

Visual Basic

```
Private Async Function GetResponse(Of T)(uri As Uri) As Task(Of T)
    Dim client As New System.Net.Http.HttpClient()
    Dim response = Await client.GetAsync(uri)

    Using stream = Await response.Content.ReadAsStreamAsync()
        Dim ser As New DataContractJsonSerializer(GetType(T))
        Return DirectCast(ser.ReadObject(stream), T)
    End Using
End Function
```

The following is an example of how you can use the **GetResponse** method.

C#

```
Uri queryUri = new Uri("[API query URL]");

//Get response from api
var r = await GetResponse<Response>(new Uri(queryUri));

//Do something with the response data
```

Visual Basic

```
Dim queryUri As Uri = New Uri("[API query URL]")

'Get response from api
Dim r As Response = Await GetResponse(Of Response)(queryUri)

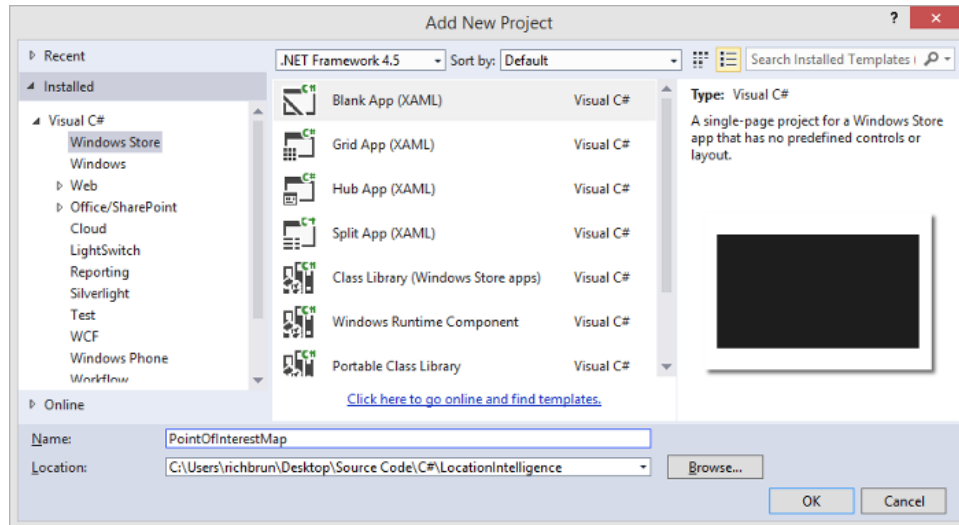
'Do something with the response data
```

Creating a Point of Interest Search app

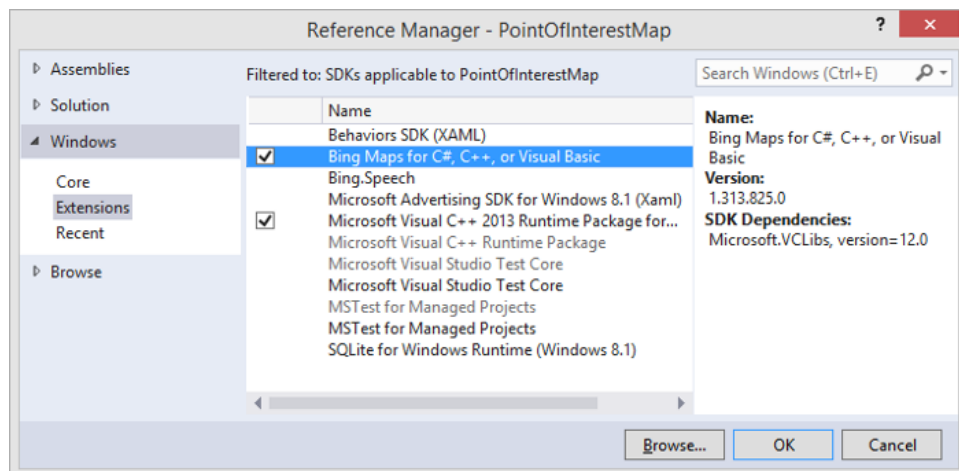
One of the most common features of a store locator app is the ability to find nearby locations. In this section we are going to use NAVTEQ point of interest data sources to create a simple app for finding nearby schools. There are millions of points of interest in the NAVTEQ data sources, to only return school locations we will need to add a filter to the query that filters the results on the **EntityTypeID** property of the NAVTEQ data source. This property can have a defined list of values which you can find documented here: <http://bit.ly/1kRCJaZ>. Schools have an entity type id of

8211. If you were building a store locator you can easily repurpose this code sample by pointing to the data source which has your store locations and updating the data contracts to align with your data source if using .NET.

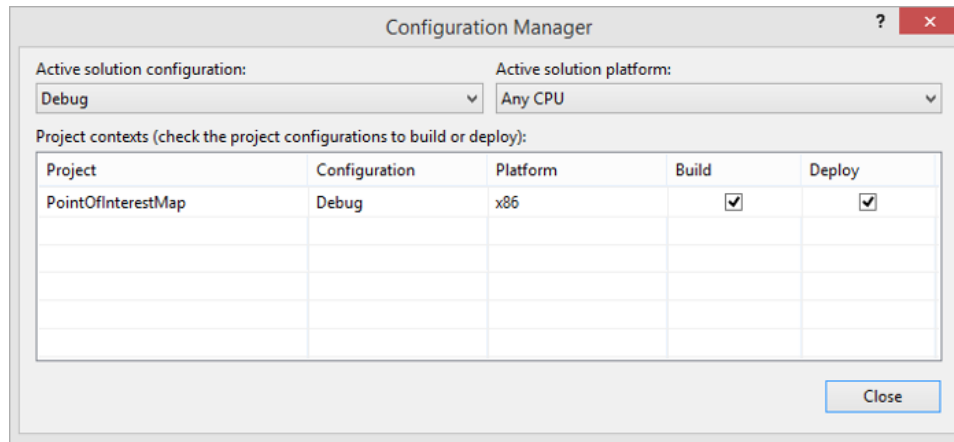
To get started open up Visual Studios and create a new project in your preferred language; JavaScript, C# or Visual Basic. Select the **Blank App** template and call the application **PointOfInterestMap** and press **OK**.



Add a reference to the Bing Maps SDK. To do this, right click on the **References** folder and press **Add Reference**. Select **Windows** -> **Extensions** select **Bing Maps for C#, C++ and Visual Basic** or **Bing Maps for JavaScript**. If you do not see this option ensure that you have installed the Bing Maps SDK for Windows Store apps. While you are here, if using C# or Visual Basic, add a reference to the **Microsoft Visual C++ Runtime Package** as this is required by the Bing Maps SDK.



If you are using C# or Visual Basic you may notice that there is a little yellow indicator on the references that you just added. The reason for this is that in the C++ runtime package you have to set the **Active solution platform** in Visual Studio to one of the following options; **ARM**, **x86** or **x64**. To do this, right click on the Solution folder and select **Properties**. Then go to **Configuration Properties -> Configuration**. Find your project and under the **Platform** column set the target platform to x86. Press OK and the yellow indicator should disappear from our references.



If you are using JavaScript right click on the **js** folder and select **Add -> New Item**. Create a new JavaScript file called **PoiMap.js**. We will put all our JavaScript for this application in there to keep things clean. Now open up the **default.html** file and update the HTML to the following.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>PointOfInterestMap</title>

  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
  <script src="//Microsoft.WinJS.2.0/js/base.js"></script>
  <script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

  <!-- PointOfInterestMap references -->
  <link href="/css/default.css" rel="stylesheet" />
  <script src="/js/default.js"></script>

  <!-- Bing Map Control references -->
  <script type="text/javascript" src="ms-
  appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>
  <script type="text/javascript" src="ms-
  appx:///Bing.Maps.JavaScript/js/veapiModules.js"></script>

  <!-- Our Bing Maps JavaScript Code -->
  <script src="/js/PoiMap.js"></script>
</head>
<body>
  <div id="myMap"></div>

  <div style="position:absolute;top:75px;right:20px;background-color:black;padding:0
  3px 0 3px;">
    <input type="text" id="searchInput" style="width:300px;" />
    <input type="button" id="searchBtn" style="font-family:Segoe UI Symbol;min-
    width:45px;" value="ⓧ" />
  </div>
</body>
</html>
```

If using C# or Visual Basic open the **MainPage.xaml** file and update it with the following.

```
<Page
  x:Class="PointOfInterestMap.MainPage"
```

```

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="using:PointOfInterestMap"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:m="using:Bing.Maps">

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <m:Map Name="MyMap" Credentials="YOUR_BING_MAPS_KEY"/>

    <StackPanel Orientation="Horizontal" Margin="0,75,20,0" Background="black"
        VerticalAlignment="Top" HorizontalAlignment="Right">
        <TextBox Name="SearchInput" Width="300"/>
        <Button Content="&#xE11A;" FontFamily="Segoe UI Symbol"
Tapped="SearchBtn_Tapped"/>
    </StackPanel>
</Grid>
</Page>

```

When the app loads we will want to load the map, get a session key from it, create a layer for adding the results and add an event handler for when the search button is clicked. We will also create a couple of global variables for storing information to make the code easier to modify. Open the **PoiMap.js** or the **MainPage.xaml.cs** file and update it with the following code.

JavaScript

```

(function () {
    var map, sessionKey, pinLayer, searchManager;

    var defaultSearchRadius = 10;
    var entityTypeId = "8211";

    function initialize() {
        Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: GetMap });

        document.getElementById("searchBtn").addEventListener("click", SearchBtn_Tapped,
true);
    }

    function GetMap() {
        map = new Microsoft.Maps.Map(document.getElementById("myMap"), {
            credentials: "YOUR_BING_MAPS_KEY"
        });

        map.getCredentials(function (c) {
            sessionKey = c;
        });

        pinLayer = new Microsoft.Maps.EntityCollection();
        map.entities.push(pinLayer);
    }

    function SearchBtn_Tapped()
    {
    }

    document.addEventListener("DOMContentLoaded", initialize, false);
})();

```

C#

```

using Bing.Maps;
using NavteqPoiSchema;
using System;
using System.Runtime.Serialization.Json;
using System.Threading.Tasks;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Input;

namespace PointOfInterestMap
{
    public sealed partial class MainPage : Page
    {
        private string sessionKey;
        private MapLayer pinLayer;

        private double defaultSearchRadius = 10;
        private string entityTypeId = "8211";

        public MainPage()
        {
            this.InitializeComponent();

            MyMap.Loaded += async (s, e) =>
            {
                try {
                    sessionKey = await MyMap.GetSessionIdAsync();

                    pinLayer = new MapLayer();
                    MyMap.Children.Add(pinLayer);
                } catch { }
            };

            private async void SearchBtn_Tapped(object sender, TappedRoutedEventArgs e)
            {
            }
        }
    }
}

```

Visual Basic

```

Imports Bing.Maps
Imports System.Runtime.Serialization.Json
Imports System.Threading.Tasks
Imports Windows.UI
Imports PointOfInterestMap.NavteqPoiSchema

Public NotInheritable Class MainPage
    Inherits Page

    Private sessionKey As String
    Private pinLayer As MapLayer

    Private defaultSearchRadius = 10
    Private entityTypeId = "8211"

    Public Sub New()

```

```

Me.InitializeComponent()

AddHandler MyMap.Loaded, Async Sub(s, e)
    Try
        sessionKey = Await MyMap.GetSessionIdAsync()

        pinLayer = New MapLayer()
        MyMap.Children.Add(pinLayer)
    Catch
    End Try
End Sub

Private Async Function SearchBtn_Tapped(sender As Object, e As
TappedRoutedEventArgs) As Task
End Function
End Class

```

If using C# or Visual Basic also add the **GetResponse** method from earlier in this chapter. Also, you will need to create the data contracts needed to deserialize the results from the NAVTEQ data source. To do this add a new class file the project called **NavteqPoiSchema**. Open this file, delete its contents and copy and paste in the following data contracts.

C#

```

using System.Runtime.Serialization;

namespace NavteqPoiSchema
{
    [DataContract]
    public class Response
    {
        [DataMember(Name = "d", EmitDefaultValue = false)]
        public ResultSet ResultSet { get; set; }
    }

    [DataContract]
    public class ResultSet
    {
        [DataMember(Name = "__copyright", EmitDefaultValue = false)]
        public string Copyright { get; set; }

        [DataMember(Name = "results", EmitDefaultValue = false)]
        public Result[] Results { get; set; }
    }

    [DataContract]
    public class Result
    {
        [DataMember(Name = "Latitude", EmitDefaultValue = false)]
        public double Latitude { get; set; }

        [DataMember(Name = "Longitude", EmitDefaultValue = false)]
        public double Longitude { get; set; }

        [DataMember(Name = "LanguageCode", EmitDefaultValue = false)]
        public string LanguageCode { get; set; }

        [DataMember(Name = "Name", EmitDefaultValue = false)]
        public string Name { get; set; }
    }
}

```



```

[DataMember(Name = "DisplayName", EmitDefaultValue = false)]
public string DisplayName { get; set; }

[DataMember(Name = "AddressLine", EmitDefaultValue = false)]
public string AddressLine { get; set; }

[DataMember(Name = "Locality", EmitDefaultValue = false)]
public string Locality { get; set; }

[DataMember(Name = "AdminDistrict", EmitDefaultValue = false)]
public string AdminDistrict { get; set; }

[DataMember(Name = "AdminDistrict2", EmitDefaultValue = false)]
public string AdminDistrict2 { get; set; }

[DataMember(Name = "PostalCode", EmitDefaultValue = false)]
public string PostalCode { get; set; }

[DataMember(Name = "CountryRegion", EmitDefaultValue = false)]
public string CountryRegion { get; set; }

[DataMember(Name = "Phone", EmitDefaultValue = false)]
public string Phone { get; set; }

[DataMember(Name = "EntityTypeID", EmitDefaultValue = false)]
public string EntityTypeID { get; set; }
}
}

```

Visual Basic

Imports System.Runtime.Serialization

Namespace NavteqPoiSchema

```

<DataContract> _
Public Class Response
    <DataMember(Name:="d", EmitDefaultValue:=False)> _
    Public Property ResultSet As ResultSet
End Class

<DataContract> _
Public Class ResultSet
    <DataMember(Name:="__copyright", EmitDefaultValue:=False)> _
    Public Property Copyright As String

    <DataMember(Name:="results", EmitDefaultValue:=False)> _
    Public Property Results As Result()
End Class

<DataContract> _
Public Class Result
    <DataMember(Name:="Latitude", EmitDefaultValue:=False)> _
    Public Property Latitude As Double

    <DataMember(Name:="Longitude", EmitDefaultValue:=False)> _
    Public Property Longitude As Double

    <DataMember(Name:="LanguageCode", EmitDefaultValue:=False)> _
    Public Property LanguageCode As String

```

```

<DataMember(Name:="Name", EmitDefaultValue:=False)> _
Public Property Name As String

<DataMember(Name:="DisplayName", EmitDefaultValue:=False)> _
Public Property DisplayName As String

<DataMember(Name:="AddressLine", EmitDefaultValue:=False)> _
Public Property AddressLine As String

<DataMember(Name:="Locality", EmitDefaultValue:=False)> _
Public Property Locality As String

<DataMember(Name:="AdminDistrict", EmitDefaultValue:=False)> _
Public Property AdminDistrict As String

<DataMember(Name:="AdminDistrict2", EmitDefaultValue:=False)> _
Public Property AdminDistrict2 As String

<DataMember(Name:="PostalCode", EmitDefaultValue:=False)> _
Public Property PostalCode As String

<DataMember(Name:="CountryRegion", EmitDefaultValue:=False)> _
Public Property CountryRegion As String

<DataMember(Name:="Phone", EmitDefaultValue:=False)> _
Public Property Phone As String

<DataMember(Name:="EntityTypeID", EmitDefaultValue:=False)> _
Public Property EntityTypeID As String
End Class
End Namespace

```

Notice that the namespace of these data contracts are set as **NavteqPoiSchema**. If you have multiple data sources you will find that the data contracts are very similar and some of the class names clash. Separating these data contracts using a unique namespace is one way to make it easier to prevent this clashing from happening.

Next we will create a couple helper methods and add them to the **PoiMap.js** and the **MainPage.xaml.cs** file. For JavaScript we will create both a **Geocode** and a **NavteqPoiSearch** methods. For C# and Visual Basic we will only create a **NavteqPoiSearch** method. The following code can be added to these files.

JavaScript

```

function Geocode(input, callback)
{
    if (searchManager) {
        var searchRequest = {
            where: input,
            callback: callback
        };
        searchManager.geocode(searchRequest);
    } else {
        //If search manager is not defined load the search module, create a
        //new instance of the search manager and call the geocode function again.
        Microsoft.Maps.loadModule('Microsoft.Maps.Search', {
            callback: function () {
                searchManager = new Microsoft.Maps.Search.SearchManager(map);
                Geocode(input, callback);
            }
        });
    }
}

```

```

    });
}
}

function NavteqPoiSearch(center, callback)
{
    var baseUrl;

    //Switch between the NAVTEQ POI data sets for NA and EU based on the longitude value.
    if (center.longitude < -30)
    {
        baseUrl =
"http://spatial.virtualearth.net/REST/v1/data/f22876ec257b474b82fe2ffcb8393150/NavteqNA/Navt
eqPOIs";
    }
    else
    {
        baseUrl =
"http://spatial.virtualearth.net/REST/v1/data/c2ae584bbccc4916a0acf75d1e6947b4/NavteqEU/Navt
eqPOIs";
    }

    var query = baseUrl + "?spatialFilter=nearby(" + center.latitude + "," +
center.longitude + "," + defaultSearchRadius + ")&$filter=EntityTypeID%20Eq%20" +
entityTypeId + "&$format=json&key=" + sessionKey;

    //Get response from NAVTEQ POI data source
    WinJS.xhr({ url: query }).then(function (e) {
        //Parse the text response into JSON
        var r = JSON.parse(e.responseText);

        callback(r);
    });
}

```

C#

```

private async Task<Response> NavteqPoiSearch(Location center)
{
    string baseUrl;

    //Switch between the NAVTEQ POI data sets for NA and EU based on the longitude value.
    if (center.Longitude < -30)
    {
        baseUrl =
"http://spatial.virtualearth.net/REST/v1/data/f22876ec257b474b82fe2ffcb8393150/NavteqNA/Navt
eqPOIs";
    }
    else
    {
        baseUrl =
"http://spatial.virtualearth.net/REST/v1/data/c2ae584bbccc4916a0acf75d1e6947b4/NavteqEU/Navt
eqPOIs";
    }

    string query =
string.Format("{0}?spatialFilter=nearby({1:N5},{2:N5},{3})&$filter=EntityTypeID%20Eq%20{4}&$
format=json&key={5}", baseUrl, center.Latitude, center.Longitude, defaultSearchRadius,
entityTypeId, sessionKey);
}

```

```

    return await GetResponse<Response>(new Uri(query));
}

```

Visual Basic

```

Private Async Function NavteqPoiSearch(center As Location) As Task(Of Response)
    Dim baseUrl As String

    'Switch between the NAVTEQ POI data sets for NA and EU based on the longitude value.
    If center.Longitude < -30 Then
        baseUrl =
"http://spatial.virtualearth.net/REST/v1/data/f22876ec257b474b82fe2ffcb8393150/NavteqNA/Navt
eqPOIs"
    Else
        baseUrl =
"http://spatial.virtualearth.net/REST/v1/data/c2ae584bbccc4916a0acf75d1e6947b4/NavteqEU/Navt
eqPOIs"
    End If

    Dim query As String =
String.Format("{0}?spatialFilter=nearby({1:N5},{2:N5},{3})&$filter=EntityTypeID%20Eq%20{4}&$
format=json&key={5}", baseUrl, center.Latitude, center.Longitude, defaultSearchRadius,
entityTypeId, sessionKey)

    Return Await GetResponse(Of Response)(New Uri(query))
End Function

```

The NAVTEQ point of interest data is separated into two data sources for better performance. The contents of these two data sources are neatly separated by the Atlantic Ocean so by doing a simple comparison on the longitude value of the search query in the **NavteqPoiSearch** method we can choose the correct data source to work with, rather than making calls to both of them. Also notice in the **NavteqPoiSearch** method that we are adding a filter on the **EntityTypeID** property of the data source.

The final step in this application is to add the logic required to carry out the search in the search button handler. When this event handler is triggered we will want to clear the map of any data, geocode the users search query, find nearby schools, display them on the map as pushpins, and then zoom into the results. Update the **SearchBtn_Tapped** event handler with the following code.

JavaScript

```

function SearchBtn_Tapped()
{
    pinLayer.clear();

    var searchInput = document.getElementById("searchInput").value;

    if (searchInput && searchInput != "")
    {
        Geocode(searchInput, function (r) {
            if (r && r.results && r.results.length > 0) {
                NavteqPoiSearch(r.results[0].location, function (poi) {
                    if (poi != null && poi.d != null &&
                        poi.d.results != null &&
                        poi.d.results.length > 0)
                    {
                        var locs = [];

```



```

    }
    }
}
catch { }
}

```

Visual Basic

```

Private Async Function SearchBtn_Tapped(sender As Object, e As TappedRoutedEventArgs) As
Task
    Try
        'Clear the pin layer
        pinLayer.Children.Clear()

        If Not String.IsNullOrEmpty(SearchInput.Text) Then
            'Geocode the search input
            Dim options = New Bing.Maps.Search.GeocodeRequestOptions(SearchInput.Text)
            Dim geocodeResult = Await MyMap.SearchManager.GeocodeAsync(options)

            If geocodeResult IsNot Nothing AndAlso geocodeResult.LocationData IsNot
Nothing AndAlso geocodeResult.LocationData.Count > 0 Then
                Dim location = geocodeResult.LocationData(0)

                'Search against the Navteq POI data set
                Dim poi = Await NavteqPoiSearch(location.Location)

                If poi IsNot Nothing AndAlso poi.ResultSet IsNot Nothing AndAlso
poi.ResultSet.Results IsNot Nothing AndAlso
poi.ResultSet.Results.Length > 0 Then

                    Dim locs = New LocationCollection()

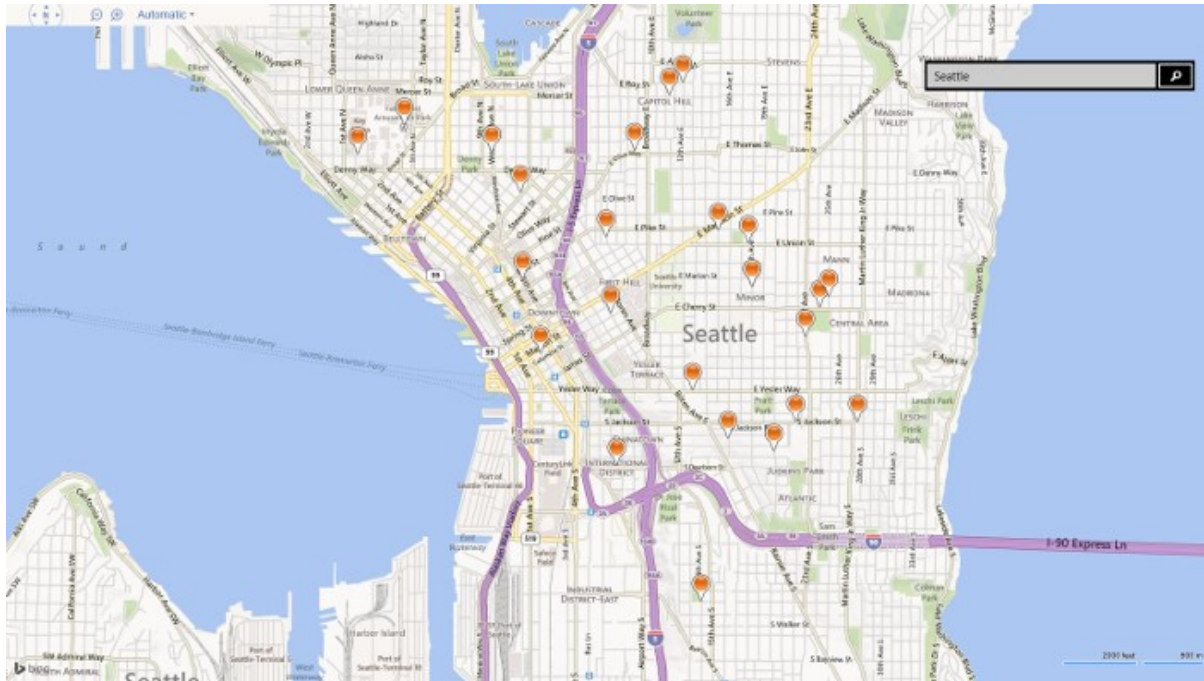
                    For Each r As Result In poi.ResultSet.Results
                        Dim loc = New Location(r.Latitude, r.Longitude)
                        locs.Add(loc)

                        Dim pin = New Pushpin()
                        MapLayer.SetPosition(pin, loc)
                        pinLayer.Children.Add(pin)
                    Next

                    MyMap.SetView(New LocationRect(locs))
                End If
            End If
        End If
    Catch
    End Try
End Function

```

At this point the application is complete. The final step is to run and test the application. Press the Debug button to launch the application. Put a location to search in the textbox and press the search button. Here is a screenshot of a search for schools in Seattle. Note that depending on which version of the Bing Maps SDK you are using you may see a different style of pushpin being displayed.



GeoData API

The GeoData API is the newest API in the Bing Spatial Data Services. In fact, at the time of writing this book this service is still marked as being in “Preview”. This API provides you access to administrative boundary data such as country and state borders. In the past if you wanted to create a map that overlaid colored polygons over well-known areas such as countries you had to locate a source for this data. Sometimes you could find a free source and other times you would have to licensing it from a data provider. In many of these cases a lot of development was required to simply connect the data to Bing Maps. This API makes this process easier by providing a number of different types of administrative boundaries that can be easily overlaid on to Bing Maps. This API has a different base URL from all the other Bing Spatial Data Services:

[http://platform.bing.com/geo/spatial/v1/public/Geodata?\[queryOptions\]&key=bingMapsKey](http://platform.bing.com/geo/spatial/v1/public/Geodata?[queryOptions]&key=bingMapsKey)

This base URL can be used to create queries to find boundary information. There are two main methods for retrieving boundaries using a spatial filter. One method lets you specify a set of coordinates that intersect with a boundary and looks like this:

spatialFilter=GetBoundary(latitude,longitude,levelOfDetail,entityType,getAllPolygons,getEntityMetadata,culture,userRegion)

The second method allows you to specify an address that intersects with the boundary and looks like this:

spatialFilter=GetBoundary(address,levelOfDetail,entityType,getAllPolygons,getEntityMetadata,culture,userRegion)

The parameters of this spatial filter are defined in the table below.

Parameter	Description
latitude, longitude	A latitude and longitude specifying a point inside the entity you are requesting. (Required)

address	An address string that is geocoded by the service to get latitude and longitude coordinates. (Required)
levelOfDetail	The level of detail for the boundary polygons returned. An integer between 0 and 3, where 0 specifies the coarsest level of boundary detail and 3 specifies the best. (Required)
entityType	A string that contains one of the following the entity types. (Required) <ul style="list-style-type: none"> • CountryRegion: Country or region • AdminDivision1: First administrative level within the country/region level, such as a state or a province. • AdminDivision2: Second administrative level within the country/region level, such as a county. • Postcode1: The smallest post code category, such as a zip code. • Postcode2: The next largest post code category after Postcode1 that is created by aggregating Postcode1 areas. • Postcode3: The next largest post code category after Postcode2 that is created by aggregating Postcode2 areas. • Postcode4: The next largest post code category after Postcode3 that is created by aggregating Postcode3 areas. • Neighborhood: A section of a populated place that is typically well-known, but often with indistinct boundaries. • PopulatedPlace: A concentrated area of human settlement, such as a city, town or village.
getAllPolygons	Specifies whether the response should include all of the boundary polygons for the requested entity or just return a single polygon that represents the main outline of the entity. (Required) <ul style="list-style-type: none"> • 0 or false: Returns only the main outline. • 1 or true: Returns all polygons.
getEntityMetadata	Specifies whether the response should include metadata about the entity, such as the area in square km of the boundary. (Required) <ul style="list-style-type: none"> • 0 or false: Do not return metadata. • 1 or true: Return all entity metadata.
culture	Specifies the preferred language to use for any metadata text about the entity or polygons. (Optional)
userRegion	Specifies the user's home country or region. A string containing a country/region code from the ISO 3166-1 alpha-2 standard. This is used to return market specific borders. (Optional)

Additionally a \$format parameter can be used as a query option to return responses as JSON or Atom XML. Since this API is in preview it's possible that the API may change when officially released. You can find the latest documentation here: <http://bit.ly/IEj69Y>

The GeoData API can be implemented in much the same way as the Query API. If using .NET the data contracts required to parse the JSON responses of the GeoData API have been included in Appendix D of this book.

GeoData Response Class

Responses from the GeoData API can be returned in JSON or Atom XML format. A response contains a result set. A result set contains an array of result objects. Depending on if you are using JSON or XML the format of the response may vary. The following URL will return the country boundary in which a coordinate intersects with as an Atom XML response.

```
https://platform.bing.com/geo/spatial/v1/public/geodata?spatialFilter=GetBoundary(47.64054,-122.12934,1,'CountryRegion',1,1,'en','us') &key=BingMapsKey
```

The response from this URL query will return the following Atom XML information.

```
<?xml version="1.0" encoding="utf-8"?>
<d:feed xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
xmlns:bsi="http://schemas.microsoft.com/bing/spatial/2010/11/odata">
  <bsi:copyright></bsi:copyright>
  <id></id>
  <entry>
    <id></id>
    <content type="application/xml">
      <m:property>
        <d:EntityID m:type="Edm.Int64">244</d:EntityID>
        <d:EntityMetadata m:type="Entity.Metadata">
          <d:BestMapViewBox>MULTIPOINT ((-172.6 18.3), (-67.4 71.7))</d:BestMapViewBox>
          <d:AreaSqKm m:type="Edm.Double">9158960</d:AreaSqKm>
          <d:OfficialCulture>en</d:OfficialCulture>
        </d:EntityMetadata>
        <d:Name m:type="Entity.Name">
          <d:EntityName>United States</d:EntityName>
          <d:Culture>en</d:Culture>
          <d:SourceID m:type="Edm.Byte">1</d:SourceID>
        </d:Name>
        <d:Primitives m:type="Collection(Entity.Primitive)">
          <d:Primitive m:type="Entity.Primitive">
            <d:PrimitiveID m:type="Edm.Int64">-7889501226</d:PrimitiveID>
            <d:Shape>1,n7pvmozyoHs01pBhws1Q_71yDpk23EtrlmC</d:Shape>
            <d:NumPoints m:type="Edm.Int64">6</d:NumPoints>
            <d:SourceID m:type="Edm.Byte">1</d:SourceID>
          </d:Primitive>
          ...
        </d:Primitives>
        <d:Copyright m:type="Entity.Copyright">
          <d:CopyrightURL></d:CopyrightURL>
          <d:Sources m:type="Collection(Entity.Copyright.Source)">
            <d:Source m:type="Entity.Copyright.Source">
              <d:SourceID m:type="Edm.Byte">1</d:SourceID>
              <d:SourceName>NVT</d:SourceName>
              <d:Copyright>© 2013 Nokia</d:Copyright>
            </d:Source>
          </d:Sources>
        </d:Copyright>
      </m:property>
    </content>
  </entry>
</d:feed>
```

This URL can be modified to return a JSON response using the **\$format** parameter as follows.

[https://platform.bing.com/geo/spatial/v1/public/geodata?spatialFilter=GetBoundary\(47.64054,-122.12934,1,'CountryRegion',1,1,'en','us'\)&\\$format=json&key=BingMapsKey](https://platform.bing.com/geo/spatial/v1/public/geodata?spatialFilter=GetBoundary(47.64054,-122.12934,1,'CountryRegion',1,1,'en','us')&$format=json&key=BingMapsKey)

The response from this URL query will return the following JSON information.

```
{
  "d": {
    "Copyright": "",
    "results": [ {
      "__metadata": {
        "uri": ""
      },
    },
  ],
}
```

```

    "EntityID":"244",
    "EntityMetadata":{
      "BestMapViewBox":"MULTIPOINT ((-172.6 18.3), (-67.4 71.7))",
      "AreaSqKm":"9158960",
      "OfficialCulture":"en"
    },
    "Name":{
      "EntityName":"United States",
      "Culture":"en",
      "SourceID":"1"
    },
    "Primitives":[{
      "PrimitiveID":"-7889501226",
      "Shape":"1,n7pvmozyoHs01pBhws1Q_71yDpk23Etr1mC",
      "NumPoints":"6",
      "SourceID":"1"
    }
    ...
  ],
  "Copyright":{
    "CopyrightURL":"",
    "Sources":[{
      "SourceID":"1",
      "SourceName":"NVT",
      "Copyright":"\u00a9 2013 Nokia"
    }]
  }
}

```

Note that some information has been removed from these response to make small enough to easily understand. Both of these responses contain a **Result** class which has the following properties.

Name	Type	Description
Copyright	Copyright	A class that contains copyright related information.
EntityID	string	A unique ID number associated with this entity.
EntityMetadata	Metadata	A collection of metadata information associated with the entity.
Name	Name	The name information for the entity.
Primitives	Primitive[]	An array of Primitive objects which contain the polygon boundary information for the entity.

The **Copyright** class is used to store information about the various data providers used to create the data in the result entity and the associated copyright information for them. This class has the following properties.

Name	Type	Description
CopyrightURL	string	A URL that lists many of the data providers for Bing Maps and their related copyright information.
Sources	CopyrightSource[]	An array of copyright information for the various sources of data used in this entity.

The **CopyrightSource** class contains the information of each individual copyright and has the following properties.

Name	Type	Description
SourceID	string	The ID of the source which aligns with the source ID used by the Primitive class.
SourceName	string	The name of the data provider represented by this Source element.

Copyright	string	The copyright string for the source.
-----------	--------	--------------------------------------

The **Metadata** class contains a number of different types of metadata which may be included with some entities. The following is a list of properties that this class has. Note that most entities will likely only have information for some and not all of these properties.

Name	Type	Description
AreaSqKm	string	The approximate total surface area (in square kilometers) covered by all the polygons that comprise this entity.
BestMapViewBox	string	An area on the Earth that provides the best map view for this entity. This area is defined as a bounding box in well-known text format.
OfficialCulture	string	The culture associated with this entity.
RegionalCulture	string	The approximate population within this entity.
PopulationClass	string	The regional culture associated with this entity.

The **Name** class may sometimes be returned for an entity and contain its well-known name. This class has the following properties.

Name	Type	Description
EntityName	string	The name associated with the entity. For example "United States"
Culture	string	The culture in which the name information is in.
SourceID	string	An ID identifying the data provider that supplied the data. This ID number will reference one of the sources listed in the CopyrightSources collection.

The **Primitive** class contains the polygon information that makes up the boundaries of the entity. This class has the following properties.

Name	Type	Description
PrimitiveID	string	A unique ID associated with this polygon primitive.
Shape	string	A comma-delimited sequence starting with the version number of the polygon set followed by a list of compressed polygon "rings" (closed paths represented by sequences of latitude and-longitude points).
NumPoints	string	The number of vertex points used to define the polygon.
SourceID	string	An ID identifying the data provider that supplied the data. This ID number will reference one of the sources listed in the CopyrightSources collection.

Decompressing a GeoData Shape

The **Shape** property in the **Primitive** class returns a comma-delimited string which contains a sequence starting with the version number of the polygon set followed by a list of compressed polygon "rings" (closed paths represented by sequences of latitude and-longitude points). This string has the following format.

[version],[compressed polygon ring 1],...,[compressed polygon ring n]

To use this string to create polygons split it on the comma character. This will create an array of strings. The first string in the array will be the version information for the polygon, skip this value and loop through the remaining strings and parse the compressed strings into a **LocationCollection** object. This parsing can be done by using this decompression algorithm can be used.

JavaScript

```
function ParseEncodedValue(value) {
```

```

    var safeCharacters =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_-";
    var list = [];
    var index = 0;
    var xsum = 0;
    var ysum = 0;
    var max = 4294967296;

    while (index < value.length) {
        var n = 0;
        var k = 0;

        while (1) {
            if (index >= value.length) {
                return null;
            }
            var b = safeCharacters.indexOf(value.charAt(index++));
            if (b == -1) {
                return null;
            }
            var tmp = ((b & 31) * (Math.pow(2, k)));

            var ht = tmp / max;
            var lt = tmp % max;

            var hn = n / max;
            var ln = n % max;

            var nl = (lt | ln) >>> 0;
            n = (ht | hn) * max + nl;
            k += 5;
            if (b < 32) break;
        }

        var diagonal = parseInt((Math.sqrt(8 * n + 5) - 1) / 2);
        n -= diagonal * (diagonal + 1) / 2;
        var ny = parseInt(n);
        var nx = diagonal - ny;
        nx = (nx >> 1) ^ -(nx & 1);
        ny = (ny >> 1) ^ -(ny & 1);
        xsum += nx;
        ysum += ny;
        var lat = ysum * 0.00001;
        var lon = xsum * 0.00001
        list.push(new Microsoft.Maps.Location(lat, lon));
    }
    return list;
}

```

C#

```

private static bool TryParseEncodedValue(string value, out LocationCollection
parsedValue)
{
    string safeCharacters =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_-";

    parsedValue = null;
    var list = new LocationCollection();
    int index = 0;

```

```

int xsum = 0, ysum = 0;

while (index < value.Length)
{
    long n = 0;
    int k = 0;

    while (true)
    {
        if (index >= value.Length)
            return false;

        int b = safeCharacters.IndexOf(value[index++]);

        if (b == -1)
            return false;

        n |= ((long)b & 31) << k;
        k += 5;
        if (b < 32) break;
    }

    int diagonal = (int)((Math.Sqrt(8 * n + 5) - 1) / 2);
    n -= diagonal * (diagonal + 1L) / 2;

    int ny = (int)n;
    int nx = diagonal - ny;

    nx = (nx >> 1) ^ -(nx & 1);
    ny = (ny >> 1) ^ -(ny & 1);

    xsum += nx;
    ysum += ny;

    double lat = ysum * 0.00001;
    double lon = xsum * 0.00001;

    //Trim latlong values to supported ranges
    lat = Math.Max(-85, Math.Min(85, lat));
    lon = Math.Max(-180, Math.Min(180, lon));

    list.Add(new Location(lat, lon));
}

parsedValue = list;
return true;
}

```

Visual Basic

```

Private Shared Function TryParseEncodedValue(value As String, ByRef parsedValue As
LocationCollection) As Boolean
    Dim safeCharacters As String =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_-"

    parsedValue = Nothing
    Dim list = New LocationCollection()
    Dim index As Integer = 0
    Dim xsum As Integer = 0, ysum As Integer = 0

```

```

While index < value.Length
    Dim n As Long = 0
    Dim k As Integer = 0

    While True
        If index >= value.Length Then
            Return False
        End If

        Dim b As Integer = safeCharacters.IndexOf(value(index))

        index += 1

        If b = -1 Then
            Return False
        End If

        n = n Or (CLng(b) And 31) << k
        k += 5
        If b < 32 Then
            Exit While
        End If
    End While

    Dim diagonal As Long = CLng(Math.Floor((Math.Sqrt(8 * n + 5) - 1) / 2))

    n = n - diagonal * (diagonal + 1) / 2

    Dim ny As Integer = CInt(n)
    Dim nx As Integer = diagonal - ny

    nx = (nx >> 1) Xor -(nx And 1)
    ny = (ny >> 1) Xor -(ny And 1)

    xsum += nx
    ysum += ny

    Dim lat As Double = ysum * 0.00001
    Dim lon As Double = xsum * 0.00001

    'Trim latlong values to supported ranges
    lat = Math.Max(-85, Math.Min(85, lat))
    lon = Math.Max(-180, Math.Min(180, lon))

    list.Add(New Location(lat, lon))
End While

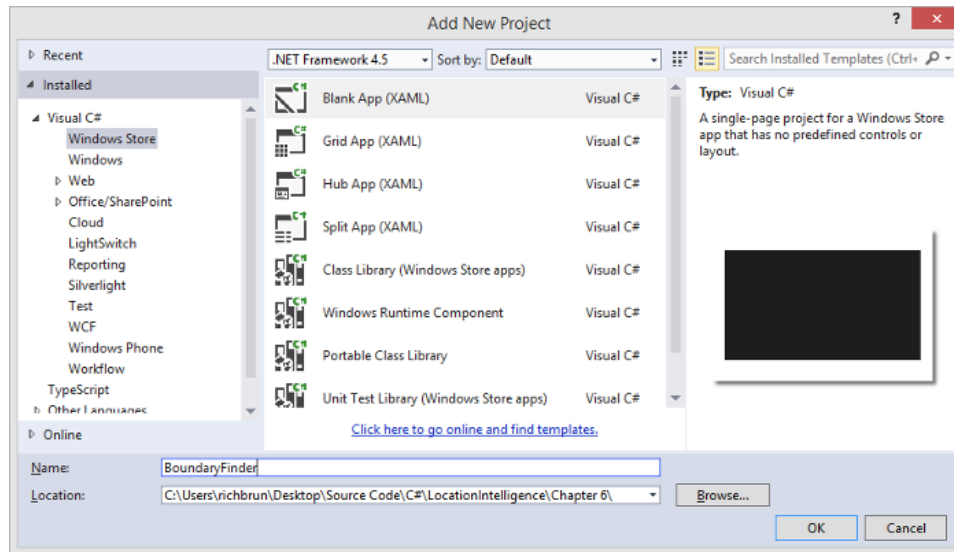
parsedValue = list
Return True
End Function

```

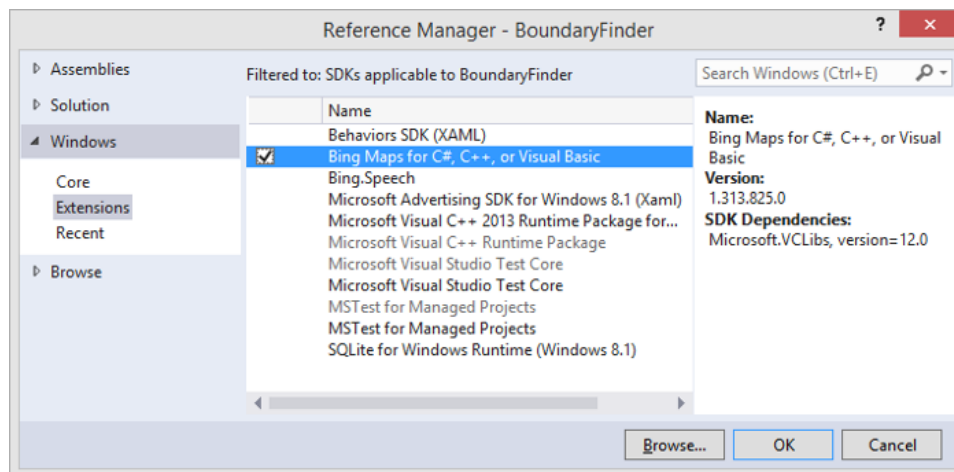
Creating a Boundary search app

When geocoding a location we can often get the **EntityType** of the location that we were searching for. Some of the **EntityType** values from the Bing Maps geocoder match those supported by the GeoData API. With this in mind we will create a simple search app that will geocode a location and display the relevant boundary from the GeoData API. For example, if we search for Chicago we will see the city boundary displayed on the map.

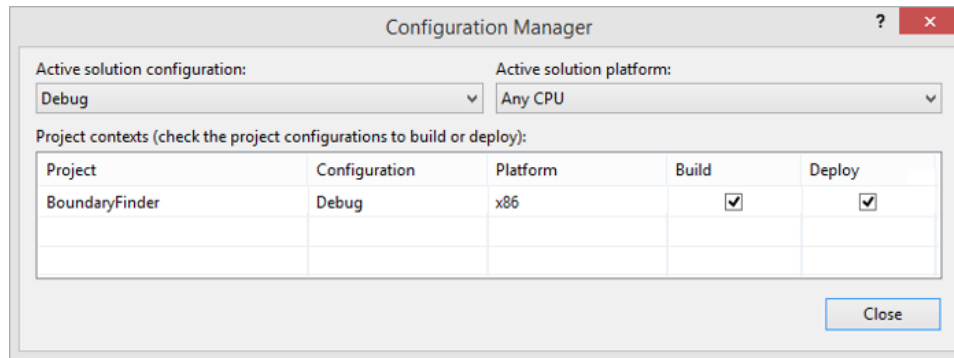
To get started open up Visual Studios and create a new project in your preferred language; JavaScript, C# or Visual Basic. Select the **Blank App** template and call the application **BoundaryFinder** and press **OK**.



Add a reference to the Bing Maps SDK. To do this, right click on the **References** folder and press **Add Reference**. Select **Windows** -> **Extensions** select **Bing Maps for C#, C++ and Visual Basic** or **Bing Maps for JavaScript**. If you do not see this option ensure that you have installed the Bing Maps SDK for Windows Store apps. While you are here, if using C# or Visual Basic, add a reference to the **Microsoft Visual C++ Runtime Package** as this is required by the Bing Maps SDK.



If you are using C# or Visual Basic you may notice that there is a little yellow indicator on the references that you just added. The reason for this is that in the C++ runtime package you have to set the **Active solution platform** in Visual Studio to one of the following options; **ARM**, **x86** or **x64**. To do this, right click on the Solution folder and select **Properties**. Then go to **Configuration Properties -> Configuration**. Find your project and under the **Platform** column set the target platform to x86. Press OK and the yellow indicator should disappear from our references.



If you are using JavaScript right click on the **js** folder and select **Add -> New Item**. Create a new JavaScript file called **BoundaryFinder.js**. We will put all our JavaScript for this application in there to keep things clean. Now open up the **default.html** file and update the HTML to the following.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>BoundaryFinder</title>

  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
  <script src="//Microsoft.WinJS.2.0/js/base.js"></script>
  <script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

  <!-- BoundaryFinder references -->
  <link href="/css/default.css" rel="stylesheet" />
  <script src="/js/default.js"></script>

  <!-- Bing Map Control references -->
  <script type="text/javascript" src="ms-
appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>
  <script type="text/javascript" src="ms-
appx:///Bing.Maps.JavaScript/js/veapiModules.js"></script>

  <!-- Our Bing Maps JavaScript Code -->
  <script src="/js/BoundaryFinder.js"></script>
</head>
<body>
  <div id="myMap"></div>

  <div style="position:absolute;top:75px;right:20px;background-color:black;padding:0
3px 0 3px 0;">
    <input type="text" id="searchInput" style="width:300px;" />
    <input type="button" id="searchBtn" style="font-family:Segoe UI Symbol;min-
width:45px;" value="#xE11A;" />
  </div>
</body>
</html>
```

If using C# or Visual Basic open the **MainPage.xaml** file and update it with the following.

```
<Page
  x:Class="BoundaryFinder.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:BoundaryFinder"
```

```

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:m="using:Bing.Maps">

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <m:Map Name="MyMap" Credentials="YOUR_BING_MAPS_KEY"/>

    <StackPanel Orientation="Horizontal" Margin="0,75,20,0" Background="black"
        VerticalAlignment="Top" HorizontalAlignment="Right">
        <TextBox Name="SearchInput" Width="300"/>
        <Button Content="&#xE11A;" FontFamily="Segoe UI Symbol"
Tapped="SearchBtn_Tapped"/>
    </StackPanel>
</Grid>
</Page>

```

When the app loads we will want to load the map, get a session key from it, create a layer for adding the boundary to and add an event handler for when the search button is clicked. Open the **BoundaryFinder.js** or the **MainPage.xaml.cs** file and update it with the following code.

JavaScript

```

(function () {
    var map, sessionKey, shapeLayer, searchManager;

    function initialize() {
        Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: GetMap });

        document.getElementById("searchBtn").addEventListener("click", SearchBtn_Tapped,
true);
    }

    function GetMap() {
        map = new Microsoft.Maps.Map(document.getElementById("myMap"), {
            credentials: "YOUR_BING_MAPS_KEY"
        });

        map.getCredentials(function (c) {
            sessionKey = c;
        });

        shapeLayer = new Microsoft.Maps.EntityCollection();
        map.entities.push(shapeLayer);
    }

    function SearchBtn_Tapped() {
    }

    document.addEventListener("DOMContentLoaded", initialize, false);
})();

```

C#

```

using Bing.Maps;
using System;
using System.Runtime.Serialization.Json;
using System.Threading.Tasks;
using Windows.UI;
using Windows.UI.Xaml.Controls;

```

```

using Windows.UI.Xaml.Input;

namespace BoundaryFinder
{
    public sealed partial class MainPage : Page
    {
        private string sessionKey;
        private MapShapeLayer shapeLayer;

        public MainPage()
        {
            this.InitializeComponent();

            MyMap.Loaded += async (s, e) =>
            {
                try
                {
                    sessionKey = await MyMap.GetSessionIdAsync();

                    shapeLayer = new MapShapeLayer();
                    MyMap.ShapeLayers.Add(shapeLayer);
                }
                catch { }
            };
        }

        private async void SearchBtn_Tapped(object sender, TappedRoutedEventArgs e)
        {
        }
    }
}

```

Visual Basic

```

Imports Bing.Maps
Imports System.Runtime.Serialization.Json
Imports System.Threading.Tasks
Imports Windows.UI

Public NotInheritable Class MainPage
    Inherits Page

    Private sessionKey As String
    Private shapeLayer As MapShapeLayer

    Public Sub New()
        Me.InitializeComponent()

        AddHandler MyMap.Loaded, Async Sub(s, e)
            Try
                sessionKey = Await MyMap.GetSessionIdAsync()

                shapeLayer = New MapShapeLayer()
                MyMap.ShapeLayers.Add(shapeLayer)
            Catch
            End Try
        End Sub

End Sub

```



```

        geoData.d.results.length > 0 &&
        geoData.d.results[0].Primitives != null &&
        geoData.d.results[0].Primitives.length > 0) {

            var strings =
geoData.d.results[0].Primitives[0].Shape.split(',');

            if (strings.length >= 2) {
                var locations = ParseEncodedValue(strings[1]);

                var polygon = new Microsoft.Maps.Polygon(locations, {
                    fillColor: new Microsoft.Maps.Color(150, 255, 0, 0)
                });
                shapeLayer.push(polygon);

                //Set the map view to show the polygon
var bounds = Microsoft.Maps.LocationRect.fromLocations(locations);
                map.setView({ bounds: bounds });
            }
        }
    });
    break;
default:
    break;
}
}
});
}
}

function RestGeocode(input, callback) {
    var geocodeUri = "http://dev.virtualearth.net/REST/v1/Locations?q=" +
encodeURIComponent(input) + "&key=" + sessionKey;

    //Get response from Bing Maps REST services
    WinJS.xhr({ url: geocodeUri }).then(function (e) {
        //Parse the text response into JSON
        var r = JSON.parse(e.responseText);

        callback(r);
    });
}

```

C#

```

private async void SearchBtn_Tapped(object sender, TappedRoutedEventArgs e)
{
    try
    {
        //Clear the shape layer
        shapeLayer.Shapes.Clear();

        if (!string.IsNullOrWhiteSpace(SearchInput.Text))
        {
            //Geocode the search input
            var options = new Bing.Maps.Search.GeocodeRequestOptions(SearchInput.Text);
            var geocodeResult = await MyMap.SearchManager.GeocodeAsync(options);

            if (geocodeResult != null &&
                geocodeResult.LocationData != null &&

```


Visual Basic

```

Private Async Function SearchBtn_Tapped(sender As Object, e As TappedRoutedEventArgs) As
Task
    Try
        'Clear the shape layer
        shapeLayer.Shapes.Clear()

        If Not String.IsNullOrEmpty(SearchInput.Text) Then
            'Geocode the search input
            Dim options = New Bing.Maps.Search.GeocodeRequestOptions(SearchInput.Text)
            Dim geocodeResult = Await MyMap.SearchManager.GeocodeAsync(options)

            If geocodeResult IsNot Nothing AndAlso geocodeResult.LocationData IsNot Nothing
            AndAlso geocodeResult.LocationData.Count > 0 Then
                Dim location = geocodeResult.LocationData(0)

                Select Case location.EntityType
                    Case "CountryRegion", "AdminDivision1", "AdminDivision2", "Postcode1",
"Postcode2", "Postcode3", _
                        "Postcode4", "Neighborhood", "PopulatedPlace"
                    Dim query =
String.Format("https://platform.bing.com/geo/spatial/v1/public/geodata?spatialFilter=GetBoun
dary({0:N5},{1:N5},0,'{2}',0,0,'en','us')&$format=json&key={3}", location.Location.Latitude,
location.Location.Longitude, location.EntityType, sessionKey)

                    Dim geoData = Await GetResponse(Of GeoDataSchema.Response)(New
Uri(query))

                    If geoData IsNot Nothing AndAlso geoData.ResultSet IsNot Nothing
            AndAlso geoData.ResultSet.Results IsNot Nothing AndAlso geoData.ResultSet.Results.Length > 0
            AndAlso geoData.ResultSet.Results(0).Primitives IsNot Nothing AndAlso
            geoData.ResultSet.Results(0).Primitives.Length > 0 Then

                        Dim strings =
geoData.ResultSet.Results(0).Primitives(0).Shape.Split(",")
                        Dim locations = New LocationCollection()

                        If strings.Length >= 2 AndAlso TryParseEncodedValue(strings(1),
locations) Then

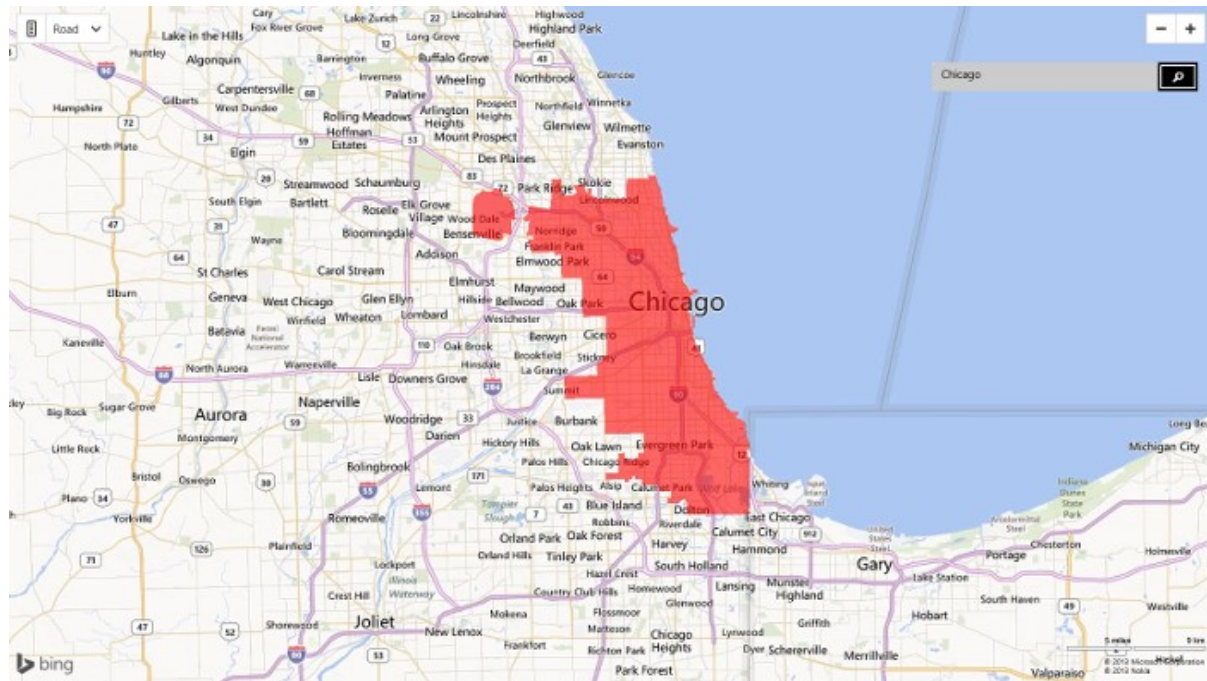
                            Dim polygon = New MapPolygon()
                            polygon.Locations = locations
                            polygon.FillColor = Color.FromArgb(150, 255, 0, 0)

                            shapeLayer.Shapes.Add(polygon)

                            'Set the map view to show the polygon
                            Dim bounds = New LocationRect(locations)
                            MyMap.SetView(bounds)
                        End If
                    End If
                Exit Select
            Case Else
                Exit Select
            End Select
        End If
    End If
    Catch
    End Try
End Function

```

At this point the application is complete. The final step is to run and test the application. Press the Debug button to launch the application. Put a location to search in the textbox and press the search button. Here is a screenshot of a search for Chicago.



Chapter Summary

In this chapter you were introduced to the Bing Spatial Data Services. Here is a summary of some of the key points.

- The Bing Spatial Data Services consists of four APIs; Geocode DataFlow, GeoData, Data Source Management, and Query.
- The Bing Maps Portal (<http://bingmapsportal.com>) is the primary place where Bing Maps developers can manage their account. Enterprise accounts have the ability to manage their data sources from within the Bing Maps portal. Documentation on how to use these tools in the Bing Maps portal can be found here: <http://bit.ly/19eVWxT>
- Trial and Basic accounts/keys can create up to 5 data sources with no more than 50 entities in each data source.
- Enterprise accounts can create up to 25 data sources with no more than 200,000 entities in each data source and no larger than 300MB in size.
- The Bing Spatial Data Services Excel add-in is an open source project available on CodePlex (<http://bsdsexceladdin.codeplex.com>) that allows you to use the batch geocode, reverse geocode and manage your data sources directly from Excel.
- The Geocode DataFlow API provides a set of tools to perform batch geocoding and reverse geocoding of large data sets. The batch process allows you to submit multiple locations to be geocoded or reverse geocoded in a single request. Full documentation this API can be found here: <http://bit.ly/1eVnM75>
- The Data Source Management API provides tools for create, edit, upload and store custom spatial data sources on the Bing Maps Servers. Full documentation on this API can be found here: <http://bit.ly/1cljBB7>

- Geometry objects can be stored in a data source or used for performing queries against data sources. Geometries must be defined in well-known text format. The standard geometry classes that are supported are **Point**, **LineString**, **Polygon**, **MultiPoint**, **MultiLineString**, **MultiPolygon**, and **GeometryCollection**.
- Bing Maps provides four public data sources which you can use in your application. These data sources consist of a sample coffee shop data source, two NAVTEQ point of interest data sources and a traffic incident data source.
- The NAVTEQ point of interest data is separated into two data sources for better performance. The contents of these two data sources are neatly separated by the Atlantic Ocean so by doing a simple comparison on the longitude value; if less than -30 use NavteqNA, otherwise use NavteqEU.
- The Query API can be used to query locations using different queries such as find nearby, find by bounding box, find along a route, intersects with a geometry, or find by property.
- When performing a nearby search the distance from the center point to each entity can be returned by specifying **__Distance** as one of the \$select query option values. This value will be in kilometers.
- The NAVTEQ point of interest data sources do not support the geometry intersect spatial filter for performance reasons.
- The GeoData API is the newest API in the Bing Spatial Data Services. This API provides you access to administrative boundary data such as country and state borders.
- The boundary data returned by the GeoData API is stored as an encoded string. This drastically reduces the response size from the service. This string will need to be decoded into a usable value before being displayed on Bing Maps.
- Many of the **EntityType** values returned by the in geocode response from Bing Maps are supported by the GeoData API. However the **EntityType** value is not returned by the **SearchManager** when using JavaScript. Use the Bing Maps REST geocoding service if using JavaScript to get these values if needed.

Chapter 7: Working with Spatial Data

A regular use case for Bing Maps is to visualize spatial data. Spatial data is the information that identifies the geographic location of features and boundaries on Earth, such as natural or constructed features, oceans, and more. Spatial data is usually stored as coordinates or as imagery, and is data that can be mapped. Spatial data is often accessed, manipulated or analyzed through Geographic Information Systems (GIS). In this chapter we are going to learn how to import many common spatial data formats into a Bing Maps Windows Store app.

Common Spatial Data Formats

The Open Geospatial Consortium (OGC) is a global organization that manages standards around geospatial data. One of their standards is a set of simple features, also referred to as a **Geometry**, which is used to define spatial objects (<http://bit.ly/H1fwH3>). The following is a list of the main simple features commonly used with Bing Maps.

Geometry Type	Description
Point	A single data point or location. This is often represented using a Pushpin in Bing Maps.
LineString	An object consisting of several connected line segments. This geometry type is similar to a Polyline in Bing Maps.
Polygon	An object consisting of several connected line segments in a closed ring. Sometimes having holes cut out.
MultiPoint	A collection of Point geometries. Useful for representing a collection of related locations. For example, this geometry type would be useful for defining the location of all ATM's in a city.
MultiLineString	An object consisting of multiple LineString 's. This is useful for representing complex paths such as river networks which may have several branches.
MultiPolygon	An object consisting of multiple polygons. This is often used to represent complex boundaries such as country boundaries that have many disconnected areas such as islands.
GeometryCollection	This is a complex geometry type that is a collection of any combination of all the other geometry types. All the geometry types that have a name starting with Multi is actually a GeometryCollection restricted to a single type of geometry.

The OGC defines other more complex Geometry types as well, however they are less commonly used than those listed above and are less likely to be used in a Bing Maps Windows Store app.

Well Known Text

Well Known Text (WKT) is an OGC standard that is used to represent spatial data in a textual format. Most OGC compliant systems support Well Known Text. The spatial functionality in SQL Server 2008, 2012 and Azure can easily convert between a spatial object in the database and WKT. A WKT can only store the information for a single spatial object and this spatial data format is usually used as part of a larger file format or web service response. The following are examples of each of the geometry types represented as Well Known Text.

Geometry Type	Well Known Text
Point	POINT(-122.349 47.651)
LineString	LINestring(-122.360 47.656, -122.343 47.656)
Polygon	POLYGON((-122.358 47.653, -122.348 47.649, -122.348 47.658, -122.358 47.658, -122.358 47.653))
MultiPoint	MULTIPOINT(-122.360 47.656, -122.343 47.656)

MultiLineString	MULTILINESTRING ((-122.358 47.653, -122.348 47.649, -122.348 47.658, -122.358 47.658, -122.358 47.653), (-122.357 47.654, -122.357 47.657, -122.349 47.657, -122.349 47.650, -122.357 47.654))
MultiPolygon	MULTIPOLYGON((((-122.358 47.653, -122.348 47.649, -122.358 47.658, -122.358 47.653)), ((-122.341 47.656, -122.341 47.661, -122.351 47.661, -122.341 47.656))))
GeometryCollection	GEOMETRYCOLLECTION (POINT(-122.34900 47.65100), LINESTRING(-122.360 47.656, -122.343 47.656))

The coordinates in a WKT shape are ordered as “longitude latitude” which is important to remember as this is the opposite convention used by Bing Maps. The reasoning for this ordering is that longitude is draw such that the values change along the x-axis and latitude along the y-axis, and as we remember from geometry class the standard convention to represent these is as “X Y”. Since WKT is nothing more than a string with a defined format they can be easily created and parsed.

Well Known Binary

Well Known Binary (WKB) is another OGC standard for representing spatial objects in a uniform manner, in this case as binary. Like WKT this spatial data format is used to store a single spatial object. This format tends to take up a smaller amount of storage space than WKT. Most spatial databases natively store spatial data in this format. Just like WKT SQL Server 2008, 2012, and Azure can easily return spatial data as Well Known Binary. A simple overview of both Well Known Text and Well Known Binary can be found on Wikipedia here: <http://bit.ly/19Rylsc>

ESRI Shapefiles

This is a binary file format created by ESRI which uses **.shp** as a file extension, and has been around for many years. ESRI is one of the largest suppliers of advance geospatial software in the world. Many of their tools are used in applications that do advance analysis against spatial data. In the past, before online mapping became as popular as it is now, many government agencies made a lot of their public spatial data available in ESRI's Shapefile format. As a result there are some excellent free sources of data available online in this format.

ESRI Shapefiles can contain point, line, polygon, and MultiPatch geometries, however only a single geometry type is used in a single file. The MultiPath geometry is a group of surfaces patches, most only maps do not support this type of geometry. A shapefile is often accompanied by additional files in different formats such as a dBASE database (**.dbf**) and projection file (**.prj**). The dBASE database is used to store metadata related to each object in the shapefile. The projection file is used to store information about the geospatial projection used to define the coordinates of the geometries in the shapefile. Bing Maps supports data that uses the WGS84 datum, if your shapefile uses a different datum you will need to reproject it. There are many free and commercial tools available to do this. One of my personal favorites is called OGR2OGR (<http://bit.ly/16fsGWK>). It is an open source command line tool that not only can be used to reproject a number of spatial file formats, but is also capable of converting spatial data from one file format to another. You can find a good blog post on how to use this tool here: <http://bit.ly/1cUDlbn>.

There are many different ways to import shapefiles into Bing Maps. If you are working on a cross platform or web based application, one of the most common approaches is to import the shapefile into one of the versions of SQL Server that support spatial data (2008, 2012, SQL Azure). Once in SQL server a web service can be easily created to expose this data to your application. The OGR2OGR tool can be used to import shapefiles into SQL Server. If you prefer not to use a command line tool then take a look at the free Shape2SQL tool (<http://bit.ly/H3AAvH>). If you plan to use this approach to access Shapefile information then take a look at these blog posts on how to create a spatial web service; <http://binged.it/1hW55zA> and <http://binged.it/19MDfRe>.

If you only need to access this file type from within a Windows Store app then there are a couple of options. One option is to import it into a SQLite database and host it locally in your application as described in this blog post: <http://binged.it/19RqH12>. Another option is to use a binary reader to parse the data from the file. Later in this chapter you will be introduced to a toolkit that provides such a parser for Windows Store apps.

KML Files

KML stands for Keyhole Markup Language and is an XML file format that is also an OGC standard. It is used for storing spatial data along with styling and view information as well. As a result these files can grow in size quickly. This file format was popular for use in online mapping applications but has since been in decline as other, more efficient formats such as GeoJSON have become more mainstream. All that said there are still plenty of existing source of data available in this format.

KML files usually use either an **.xml** or **.kml** file extension. There is a compressed version of KML which uses a **.kmz** file extension. If you unzip a KMZ file you will find a KML file and sometimes additional assets such as images inside. The following is an example of a KML file containing the location of New York.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <Placemark>
      <name>New York City</name>
      <description>New York City</description>
      <Point>
        <coordinates>-74.006393,40.714172,0</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

Additional information on this file format can be found here: <http://bit.ly/1csSR0C>

GeoRSS Files

A GeoRSS file is an XML based web standard for embedding geospatial information inside of a RSS or Atom feed. These files typically use a file extension of **.xml** or **.rss**. There are two versions of the GeoRSS format; simple and GML. The simple format supports points, lines, and polygons. The polygons in this format only support exterior rings and have no support for interior rings or holes. The GML version of this file format has support for all of the OGC geometry types. The following is an example of a GeoRSS file containing the location of New York.

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0"
  xmlns:georss="http://www.georss.org/georss"
  xmlns:gml="http://www.opengis.net/gml">
  <channel>
    <item>
      <name>New York City</name>
      <description>New York City</description>
      <georss:point>40.714172 -74.006393</georss:point>
    </item>
  </channel>
</rss>
```

Additional information on this file format can be found here: <http://bit.ly/H3Hhh6>

GPX Files

GPX is yet another common XML format used for storing spatial data. This file type typically use an **.xml** or **.gpx** file extension. This file format is commonly used by GPS devices and as such has become very popular. It can be used to describe waypoints, tracks, and routes. Chances are, if you have a standalone GPS device it is capable of importing and exporting information in this format. Viewing this data on a GPS device is great when you are outdoors, but when you have access to a computer it is usually much easier overlay these files over maps to see where you have been. The following is an example of a GPX file containing the location of New York as a waypoint.

```

<?xml version="1.0"?>
<gpx
  version="1.0"
  creator=""
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.topografix.com/GPX/1/0"
  xsi:schemaLocation="http://www.topografix.com/GPX/1/0
http://www.topografix.com/GPX/1/0/gpx.xsd">
  <wpt lat="40.714172" lon="-74.006393">
    <name>New York</name>
    <desc>New York</desc>
  </wpt>
</gpx>

```

Additional information on this file format can be found here: <http://bit.ly/1hWbUkx>

GeoJSON Format

GeoJSON is a relatively new file format used for storing spatial data. When storing in a file the **.js** or **.json** file extensions are used. This file format tends to use a lot less characters than its XML equivalents. This results in a much smaller file size, making it ideal for transferring spatial data to web and mobile applications. The following is an example of a GeoJSON file containing the location of New York.

```

{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [-74.006393, 40.714172]
      },
      "properties": {
        "name": "New York",
        "description": "New York"
      }
    }
  ]
}

```

Additional information on this file format can be found here: <http://bit.ly/1aN5Ub7>

Importing Data using JavaScript Modules

The Bing Maps Windows Store JavaScript SDK is based on the Bing Maps v7 AJAX SDK which has been available for online mapping application for many years. In that time a number of modules have been created to make it easy to import many common spatial data formats. These modules are ideal if you are creating a cross platform application using JavaScript as you will be able to not only use them in your Windows Store app but in any browser that Bing Maps supports. All of these modules and many more are available through the open source Bing Maps V7 Modules project on CodePlex (<http://bingmapsv7modules.codeplex.com/>). Here is a list of the current modules available for importing spatial data into Bing Maps.

Module Name	Link
GeoJSON Module	http://bit.ly/19Ml6mE
GeoRSS Module	http://bit.ly/1cs6rS3
GPX Parser	http://bit.ly/GXMRSW

Well Known Text Reader/Writer	http://bit.ly/1aWMgdU
-------------------------------	---

Each module contains examples of how to implement them in Bing Maps. Later in this chapter we will go into more detail about each of these spatial data formats.

Implementing the Well Known Text Module

Well Known Text (WKT) is a common method used for representing spatial objects as a string as we saw earlier in this chapter. The main purpose of this section is to show how to implement the Well Known Text modules in a JavaScript Windows Store app.

To implement the Well Known Text module you first need to download it from the link provided earlier. Create a new project in Visual Studio called **WellKnownText** using the blank app template and add the **WKTModule.js** file to the **js** folder. Create a JavaScript file in the **js** folder called **WktSample.js** where the application logic can be added. Add a reference to the Bing Maps SDK. To do this right click on the **References** folder and press **Add Reference**. Select **Windows -> Extensions**, and then select **Bing Maps for JavaScript**.

There are three steps involved in implementing modules:

1. Add a reference to the **veapicore.js** and **veapimodules.js** files from the Bing Maps SDK.
2. Register the module if it isn't already registered.
3. Load the module and run any post load logic.

To add a reference to the **veapicore.js** and **veapimodules.js** files from the Bing Maps SDK, open the **default.html** file and add the following script references:

```
<script type="text/javascript"
    src="ms-appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>
<script type="text/javascript"
    src="ms-appx:///Bing.Maps.JavaScript/js/veapiModules.js"></script>
```

All modules built into Bing Maps are already registered. Custom module, such as the ones from the CodePlex project, will need to be registered. If you have used custom modules before, be warned, registering modules in the web version of Bing Maps is done differently from the Windows Store App version. Registering custom modules is pretty straightforward; simply add a line of code to register the name and then load the JavaScript file containing your module code right after it. The following code can be added to the **default.html** page to register the WKT module.

```
<script>Microsoft.Maps.registerModule('WKTModule');</script>
<script type="text/javascript" src="/js/WKTModule.js"></script>
```

While in the **default.html** file add a script reference to the **WktSample.js** file. The layout of the page will be simple and consist of a full screen map with a panel in the top right corner that has a text area for inserting WKT and a button to parse the text and display it on the map. Update the **default.html** file to look like this.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Well Known Text</title>

    <!-- WinJS references -->
    <link href="//Microsoft.WinJS.1.0/css/ui-dark.css" rel="stylesheet" />
    <script src="//Microsoft.WinJS.1.0/js/base.js"></script>
    <script src="//Microsoft.WinJS.1.0/js/ui.js"></script>

    <!-- OverlayingData references -->
    <link href="/css/default.css" rel="stylesheet" />
```

```

<script src="/js/default.js"></script>

<!-- Bing Map Control references -->
<script type="text/javascript" src="ms-
appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>
<script type="text/javascript" src="ms-
appx:///Bing.Maps.JavaScript/js/veapiModules.js"></script>

<!-- Load the WKT Module -->
<script>Microsoft.Maps.registerModule('WKTModule');

```

Now open the **WktSample.js** file. In this file add the logic for loading the map and the WKT module. A click event for the WKT import button will also need to be created. When the button is clicked the WKT will be copied from the text area and passed to the WKT module **Read** method. This will return a Bing Maps shape object that will be a **Pushpin**, **Polyline**, **Polygon**, or an **EntityCollection** containing any combination of these base shapes. The returned shape can be added directly to the map. Add the following code to the **WktSample.js** file.

```

(function () {
  "use strict";

  var map;

  function GetMap() {
    map = new Microsoft.Maps.Map(document.getElementById("MyMap"), {
      credentials: "YOUR_BING_MAPS_KEY"
    });

    document.getElementById('ReadWktBtn').onclick = ReadWKT;

    Microsoft.Maps.loadModule('WKTModule');
  }

  function ReadWKT() {
    var wkt = document.getElementById('WktInput').value;
    var shape = WKTModule.Read(wkt);

    if (shape != null) {
      map.entities.clear();
      map.entities.push(shape);
    } else {
      new Windows.UI.Popups.MessageDialog('Unable to parse WKT.').showAsync();
    }
  }
}

```

```

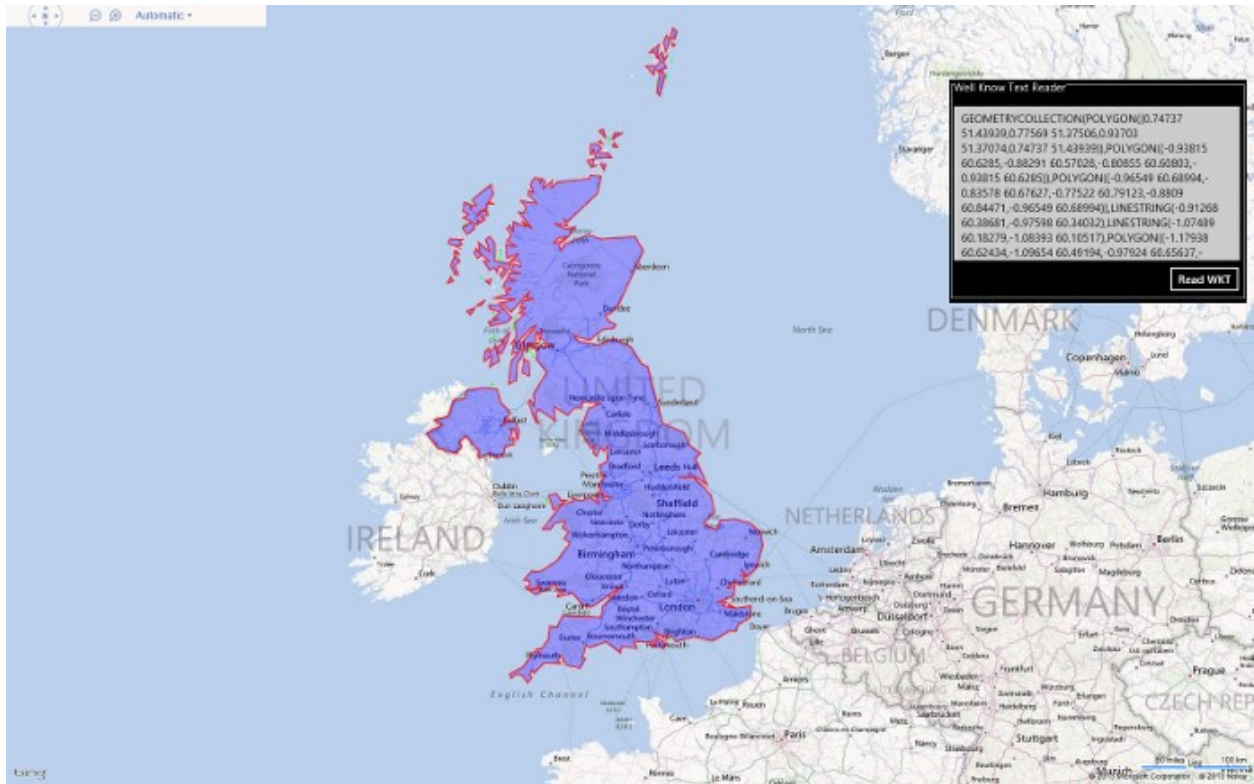
}

function initialize() {
    Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: GetMap });
}

document.addEventListener("DOMContentLoaded", initialize, false);
})();

```

Running this app and passing in a WKT value of "POINT(-0.12714 51.50632)" will display a pushpin on top of London, UK. Taking this a bit further here is a screenshot of a large WKT value of the UK boundaries being displayed on the map.



Tip: If you have an SQL 2008, 2012, or Azure database that contains some spatial data you can easily use the **STAsText** or **ToString** methods in SQL to see the Well Known text version of the spatial data. You can then copy the well-known text into this app to see what it looks like on Bing Maps.

Introduction to the Microsoft Maps Spatial Toolbox Library

As we saw in the previous section there are a number of tools available for importing spatial data into the JavaScript version of Bing Maps. Unfortunately when I started writing this book there were no equivalent tools available for native Bing Maps Windows Store apps. However, I have written a lot of tools for importing different types of spatial data into previous versions of Bing Maps, namely for Silverlight and WPF. In preparation of writing this chapter I pulled all of these tools together to create a single reusable library that not only works in Windows Store apps, but also supports Windows Phone 8, and WPF applications as well. This library is called the Microsoft Maps Spatial Toolbox. This library is used throughout the code samples in this chapter and is also available as an open source project on CodePlex here: <https://mapstoolbox.codeplex.com/>.

At the root of this toolkit is a core library of simple feature classes as defined by the Open Geospatial Consortium (OGC). These simple feature classes consist of **Point**, **LineString**, **Polygon**, **MultiPoint**, **MultiLineString**, **MultiPolygon**, and **GeometryCollection**. All of these classes derive from an abstract class called **Geometry**. These **Geometry** classes also make use of the following classes available in the library; **Coordinate**, **CoordinateCollection**, and **BoundingBox**. These are very similar to the Bing Maps classes; **Location**, **LocationCollection**, and **LocationRect**. The reason for creating new classes for representing this type of information is that the Bing Maps classes all inherit from the **DependencyObject** class. New instances of these classes can only be created on the UI thread. The geometry version of these classes can be used on any thread, thus giving us the ability to use them in background tasks to free up the UI thread when doing calculations. This also allows us to separate the geometry classes in a Portable Class Library which can be used across multiple frameworks. We will talk more about Portable Class Libraries in Chapter 11.

In addition to the geometry classes the core library also has a **DistanceUnits** enumerator and a static class called **SpatialTools**. The **SpatialTools** class contains a number of useful spatial calculations. Here is a list of the various spatial calculation methods available in this class.

Name	Description
CalculateBearing	Calculates the bearing (heading) from one coordinate to another.
CalculateDestinationCoordinate	Calculates a destination coordinate based on a starting coordinate, a bearing, a distance, and a distance unit type.
CalculateGeodesic	Takes a list of coordinates and fills in the space between them with accurately positioned points to form a Geodesic path.
CalculateMidpoint	Calculates the midpoint coordinate between two coordinates.
ConvertDistance	Converts a distance from one distance unit to another.
DecimalDegreeToDMS	Converts a decimal degree value into a string in the format of "days, minutes, seconds".
DMSToDecimalDegree	Converts a "days, minutes, seconds" value into a decimal degree value.
GenerateCircle	Calculates a list of coordinates that are an equal distance away from a central point to create an approximate circle.
GenerateRegularPolygon	Calculates a specified number of coordinates that are an equal distance away from a central point to create a regular polygon.
GetEarthRadius	Gets the approximate radius of the earth in Kilometers, Meters, Miles, and Feet.
HaversineDistance	Calculates the distance between two coordinates on the surface of a sphere (Earth).
ToDegrees	Converts an angle that is in radians to degrees.
ToRadians	Converts an angle that is in degrees to radians.
VertexReduction	Vertex reduction is the brute-force algorithm for simplifying a list of coordinates. It does this by ensuring that no two coordinates are closer than a specified distance in the unit of measurement. If working with coordinates in degrees then the distance would be in degrees.

This library includes a number of tools for reading and writing common spatial data formats. These tools are located under the **Microsoft.Maps.SpatialToolbox.IO** namespace and derive from a common abstract class called **BaseFeed**. Classes that derive from this format all have methods for asynchronously reading and writing spatial data to and from a stream. The following is a table of all the tools available in this library for reading and writing spatial data.

Spatial File Format	File Extension	BaseFeed Class Name	Read Methods	Write Methods
Well Known Text	n/a	WellKnownText	✓	✓
Well Known Binary	n/a	WellKnownBinary	✓	✓
ESRI Shapefile	.shp	ShapefileReader	✓	X
GeoRSS	.xml, .rss	GeoRSSFeed	✓	✓
GPX	.xml, .gpx	GpxFeed	✓	✓

KML	.xml, .kml	KmlFeed	✓	✓
KMZ	.zip, .kmz	KmlFeed	✓	✓
GeoJSON	.js, .json	GeoJsonFeed	✓	✓

All of these tools can read data from the specified spatial file formats, however writing is currently supported for all but shapefiles. The read and write methods of these classes make use of a class called **SpatialDataSet**. This class is used to store a group of geometry objects along with their associated metadata and style information. The **SpatialDataSet** class has the following properties:

Name	Description
BoundingBox	The bounding box of the data set.
Error	A string with an error value. Often used if there is an issue parsing a file into a spatial data set.
Geometries	A list of geometries in the data set. All of these geometries derive from the Geometry class.
Metadata	Is a ShapeMetadata class that has three properties; Title , Description and Properties . The Title and Description properties are strings. The Properties property is a Dictionary of string and objects used to store additional metadata. All geometry objects have a Metadata property as well.
Styles	A dictionary used as a lookup table for styles. Style information is stored in a ShapeStyle class.

In addition to the core library there are a number of additional libraries available that target specific platforms. The **Microsoft.Maps.SpatialToolbox.Win8** library has additional tools available that are specific to Windows Store Apps. In this library is a set of tools for the Bing Maps Windows Store SDK under the **Microsoft.Maps.SpatialToolbox.Bing** namespace. At the root of this namespace are seven classes that add Bing Maps specific features to the library. Here is an overview of these classes.

Class Name	Description
BingExtensions	A static class used to extend the Bing Maps classes to provide additional functionality. This class adds a method called ToBMGeometry to all geometries for easily converting Geometry objects to a Bing Maps shape. This class also add a method called ToGeometry to all Bing Maps shapes.
HeatMapLayer	A control that can be added as a child of the map to generate density heat maps from point based data.
MapShapeExt	A static class which provides a DependencyProperty that exposes a Tag property on the MapPolygon and MapPolyline classes.
MapTools	A static class with methods that makes it easy to add a SpatialDataSet or a Geometry object to the map.
PointCompression	This class contains the algorithms required to encode and decode coordinates that are compressed as a 64 bit string. This is used by the Bing Maps REST services for passing a large number of coordinates to the Elevation service. This is also used by the GeoData API in the Bing Spatial Data Services for passing boundary information in a compressed format.
StyleTools	This class contains a number of methods for creating Bing Maps Shapes from geometries using styles from a ShapeStyle class.
TileMath	A set of mathematical calculation based on the quad key tile system used in Bing Maps. This is handy when creating advance mapping applications.

The Pushpin class in Bing Maps has a property called **Tag**. This property allows us to store any object in it. This is handy for linking a pushpin to its associated metadata. Unfortunately the **MapPolygon** and **MapPolyline** classes do not have this property. The **MapShapeExt** class provides a **DependencyProperty** that exposes a **Tag** property on the **MapPolygon** and **MapPolyline** classes. This **Tag** property can be used to store and retrieve an **object** from a **MapPolygon** or **MapPolyline** as shown in the following example.

C#

```
MapPolygon p = new MapPolygon();
p.SetValue(MapShapeExt.TagProperty, "My Metedata");
object tagValue = p.GetValue(MapShapeExt.TagProperty);
```

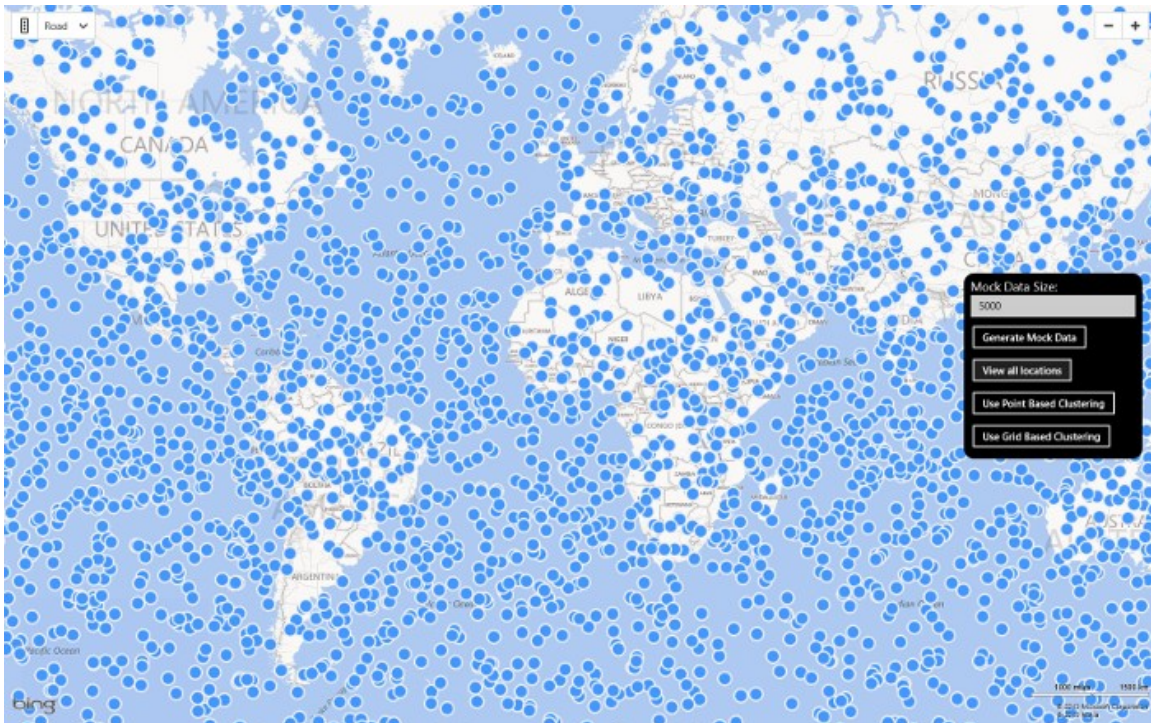
Visual Basic

```
Dim p = New MapPolygon()
p.SetValue(MapShapeExt.TagProperty, "My Metedata")
Dim tagValue = p.GetValue(MapShapeExt.TagProperty)
```

In the Bing set of tools are two sub sections of functionality; Services and Clustering. The services section consists of the data contracts needed to deserialize JSON responses from the Bing Maps REST Services, GeoData API, and the NAVTEQ point of interest data in the Bing Spatial Data Services. The Bing Maps REST Service data contracts are the same as those used in Chapter 5, however using a different namespace inside this library.

Clustering Tools

The clustering section of this library provides a set of tools for implementing location clustering. Clustering of locations in Bing Maps consists of grouping together nearby locations into clusters. As the user zooms in the clusters break apart to reveal the individual locations. The goal of this process is to reduce the number of pushpins that are displayed on the map at any given time. This results in better performance by the map control and also a better experience for the user as they will be able to see the map and not be overwhelmed with pushpins filling the screen. By implementing clustering in the native version of Bing Maps you can easily increase the number of pushpins you can add to the map before any noticeable performance issue occurs from a couple thousand to 30,000 or more. Here is a screenshot of 5,000 pushpins on a map.



Now here is the same 5,000 pushpins being displayed using clustering.



The clusters are shown in red to make them easier to see. The blue pushpins are individual locations that had no other pushpins close by.

There are several different algorithms that can be used to cluster pushpins. In this library I have provided two methods; Grid and Point based clustering. Grid based clustering breaks up the current viewport into a grid of a defined size. All pushpins inside of a single grid cell are grouped together as a cluster. The cluster is then displayed on the map at the mean average location of all the items in the cluster as this looks the most natural. However this can result in some clusters overlapping if there are a number of pushpins near the edge of a grid cell. This algorithm is very fast and ideal for large data sets upwards of 50,000 locations. However, whenever the map is moved, even slightly, a new grid is created based on the new viewport and the data re-clustered.

The point base algorithm is a bit more complex and starts off by taking the first location in the data set and searches the data set for all locations that are within a certain radius of that location. The algorithm then grabs the next, un-clustered location in the data set and searches the list again. This repeats until all items in the data set have been clustered. This extra complexity comes at a cost and I have found it works well with data sets containing up to 30,000 locations. One benefit of this approach is that clusters will rarely if ever overlap.

The clustering functionality in this library consists of a number of different classes as described here.

Name	Description
ClusteredPoint	An object used to store information about a cluster.
ClusteringLayer	A layer that can be added to the map like a MapLayer that implements the clustering logic to a data set.
ClusteringType	An enumerator which is used to specify the type of clustering algorithm to be used by the ClusteringLayer .
ItemLocation	A class used to represent a single data point to be clustered.
ItemLocationCollection	A class that represents a collection of items to be clustered by the ClusteringLayer .

The **ClusteringLayer** is the main class that does all the heavy lifting. This class can be added to the map just like a **MapLayer**. This class has three public properties as defined below.

Name	Description
ClusterRadius	The pixel radius to use in the clustering algorithm. The smaller the radius the slower the algorithms become but the more natural the data looks.
ClusterType	An enumerator of type ClusteringType which indicates the type of clustering algorithm to used; Grid or Point based clustering.
Items	A collection of location items to cluster.

Tip: It is best to first load all items into a **LocationItemCollection** object first and then add them to the **ClusterLayer Items** property using the **AddRange** method as the **Items** property will force a recluster of the data every time it's data changes.

The **ClusterLayer** class also has two events; **CreateItemPushpin** and **CreateClusteredItemPushpin**. These events allow you to specify callback methods that are used to generate the pushpins needed to represent a single or clustered location.

The following is an example of how to implement the **ClusteringLayer**.

C#

```
using Microsoft.Maps.SpatialToolbox.Bing.Clustering;

private void ClusterData_Clicked(object sender, RoutedEventArgs e)
{
    //Create an instance of the clustering layer
    var layer = new ClusteringLayer();
    layer.ClusterType = ClusteringType.Point;

    //Add event handlers to create the pushpins
    layer.CreateItemPushpin += CreateItemPushpin;
    layer.CreateClusteredItemPushpin += CreateClusteredItemPushpin;

    //Add data to layer, where _mockdata is a ItemLocationCollection
    layer.Items.AddRange(_mockData);
    MyMap.Children.Add(layer);
}

private UIElement CreateItemPushpin(object item)
{
    var pin = new Pushpin()
    {
        Tag = item
    };

    pin.Tapped += pin_Tapped;

    return pin;
}

private UIElement CreateClusteredItemPushpin(ClusteredPoint clusterInfo)
{
    var pin = new Pushpin()
    {
        Background = new SolidColorBrush(Colors.Red),
        Text = "+",
        Tag = clusterInfo
    };

    pin.Tapped += pin_Tapped;
}
```



```

    return pin;
}

```

Visual Basic

```

Imports Microsoft.Maps.SpatialToolbox.Bing.Clustering

Private Sub PointClusterData_Clicked(sender As Object, e As RoutedEventArgs)
    'Create an instance of clustering layer
    Dim layer = New ClusteringLayer()
    layer.ClusterType = ClusteringType.Point

    'Add event handlers to create the pushpins
    AddHandler layer.CreateItemPushpin, AddressOf CreateItemPushpin
    AddHandler layer.CreateClusteredItemPushpin, AddressOf CreateClusteredItemPushpin

    ' Add data to layer, where _mockdata is a ItemLocationCollection
    layer.Items.AddRange(_mockData)
    MyMap.Children.Add(layer)
End Sub

Private Function CreateItemPushpin(item As Object) As UIElement
    Dim pin = New Pushpin()
    pin.Tag = item
    Return pin
End Function

Private Function CreateClusteredItemPushpin(clusterInfo As ClusteredPoint) As UIElement
    Dim pin = New Pushpin()
    pin.Background = New SolidColorBrush(Colors.Red)
    pin.Text = "+"
    pin.Tag = clusterInfo
    Return pin
End Function

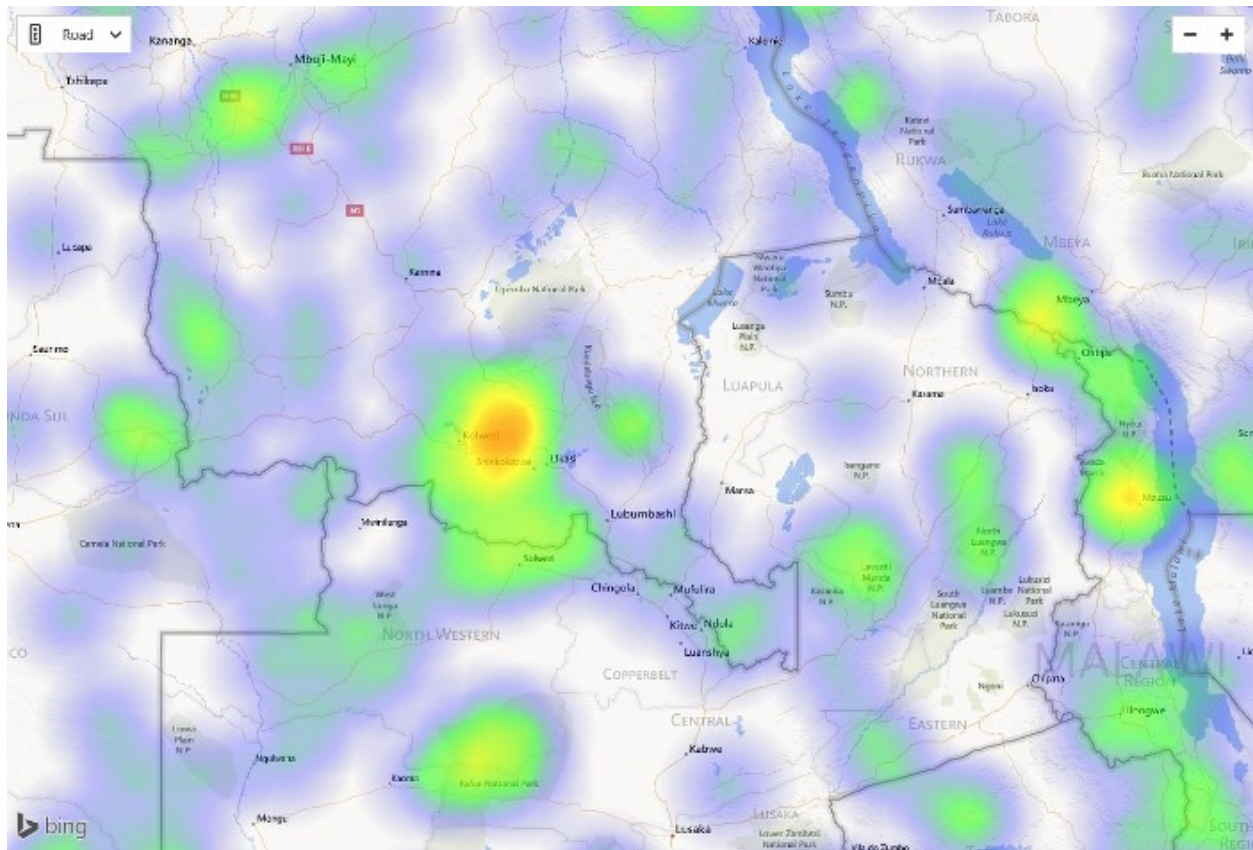
```

If you are working on a cross platform JavaScript application it is worth noting that both the Grid and Point based clustering algorithms are available as modules through the Bing Maps Modules CodePlex site. You can find these modules here:

- Grid based clustering module: <http://bit.ly/17Pkblj>
- Point based clustering module: <http://bit.ly/1cfxoH6>

Heat Map Layer

Heat maps, also known as Choropleth maps, is a type of overlay on a map used to represent data using different colors. Heat maps are often used to show the data hot spots on the map. The data used in these overlays usually takes one of two forms. The first form is color coded polygons or polylines based on some metric related to those shapes. Creating this type of heat map is fairly easy as you simply need to assign a color to the shape when adding it to the map. The second form uses point based data, the colors of the heat map are based on the density of the data points on the map. Creating these types of heat maps are a bit more complex. To simplify the creation of density heat maps the spatial toolbox has a class called **HeatMapLayer** which is inside the **Microsoft.Maps.SpatialToolbox.Bing** namespace. Here is an example of the type of heat map this class generates.



The **HeatMapLayer** class provides a number of properties which can be used to customize how the heat map is rendered. Here is a list of the different properties available.

Name	Type	Description
HeatGradient	GradientStopCollection	A color gradient used to colorize the heat map.
Intensity	double	Intensity of the heat map. A value between 0 and 1.
Locations	LocationCollection	Collection of locations to plot on the heat map.
ParentMap	Map	A reference to the parent Bing Maps control.
Radius	double	Radius of data point in meters.
EnableHardEdge	bool	Gives all values the same opacity to create a hard edge on each data point. When set to false (default) the data points will use a fading opacity towards the edges.

The heat map will not render until the **ParentMap** property is assigned. All the properties with the exception of this one support data binding. If you decided to add a heat map to your map using XAML you will need to set the **ParentMap** property in your code behind. Here is an example of how to add a heat map to your map using XAML.

```
<Page
    x:Class="HeatMapTestApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:HeatMapTestApp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:m="using:Bing.Maps"
    xmlns:hm="using:Microsoft.Maps.SpatialToolbox.Bing">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
```



```

        layer.ParentMap = MyMap
        layer.Locations = locs
    End Sub
End Class

```

Alternatively you can create the heat map completely from code and add it as a child of the map. The following is an example of how to add a **HeatMapLayer** to Bing Maps using code.

C#

```

using Bing.Maps;
using Microsoft.Maps.SpatialToolbox;
using Microsoft.Maps.SpatialToolbox.Bing;
using System;
using Windows.UI.Xaml.Controls;

namespace HeatMapTestApp
{
    public sealed partial class MainPage : Page
    {
        private HeatMapLayer layer;
        private LocationCollection locs;

        public MainPage()
        {
            this.InitializeComponent();

            this.Loaded += (s, e) =>
            {
                locs = new LocationCollection();

                //Add location data to collection...

                layer = new HeatMapLayer()
                {
                    ParentMap = MyMap,
                    Locations = locs,
                    Radius = 2500
                };

                MyMap.Children.Add(layer);
            };
        }
    }
}

```

Visual Basic

```

Imports Bing.Maps
Imports Microsoft.Maps.SpatialToolbox
Imports Microsoft.Maps.SpatialToolbox.Bing

Public NotInheritable Class MainPage
    Inherits Page

    Private layer As HeatMapLayer
    Private locs As LocationCollection

```

```

Public Sub New()
    InitializeComponent()

    AddHandler MyMap.Loaded, Sub()
        locs = New LocationCollection()

        'Add location data to collection...

        layer = New HeatMapLayer()
        layer.ParentMap = MyMap
        layer.Locations = locs
        layer.Radius = 2500

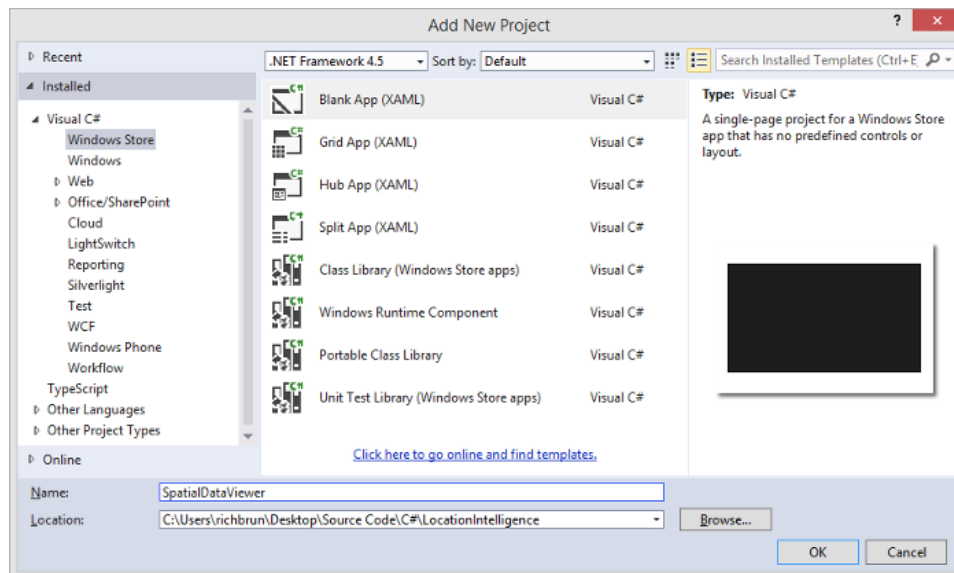
        MyMap.Children.Add(layer)
    End Sub
End Sub
End Class

```

If you are working with JavaScript there is a client side heat map module available in the Bing Maps V7 Modules project (<https://bingmapsv7modules.codeplex.com/>).

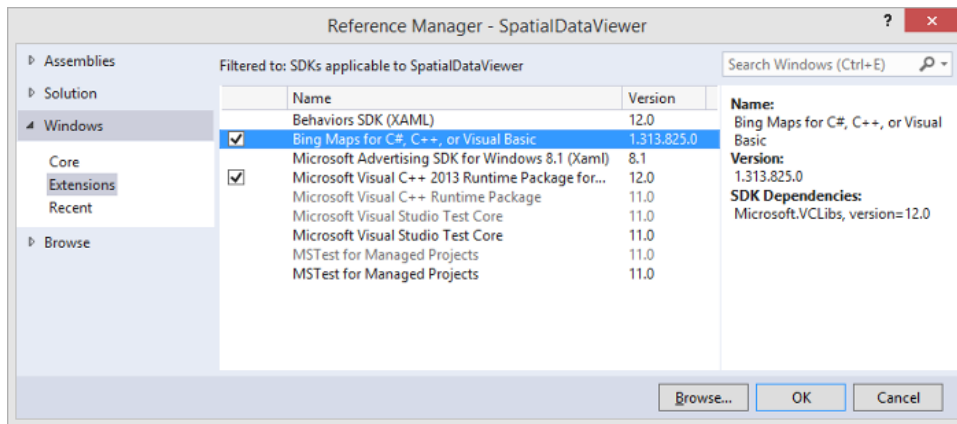
Creating a Spatial Data Viewer in .NET

In this section we are going to create a simple application that allows us to easily import common spatial data files into a Bing Maps Windows Store app. To get started, open up Visual Studios and create a new project in your preferred language: C# or Visual Basic. Select the **Blank App** template and call the application **SpatialDataViewer** and press **OK**.



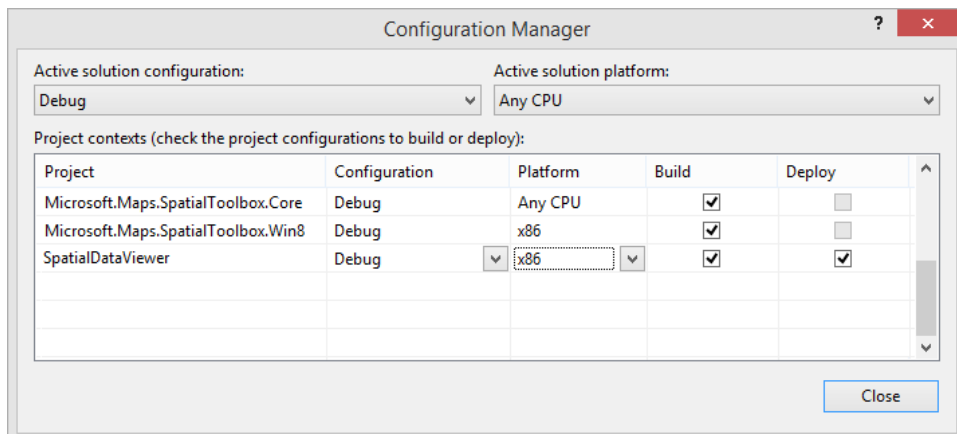
Next get a copy of the spatial toolbox library, either copy it from the code samples for this book or download it from the CodePlex site (<http://mapstoolbox.codeplex.com/>). Navigate to the folder where the **SpatialDataViewer** project is located and add a copy of the **Microsoft.Maps.SpatialToolbox.Core** and **Microsoft.Maps.SpatialToolbox.Win8** projects to that folder. Add these projects to the solution by right clicking on the solution folder in Visual Studios and selecting **Add -> Existing Project**. Next add references to these projects in the **SpatialDataViewer** by right clicking on the **References** folder and pressing **Add Reference**. Select **Solution -> Projects**, and then select the

Microsoft.Maps.SpatialToolbox.Core and **Microsoft.Maps.SpatialToolbox.Win8** projects. While you are here add a reference to the Bing Maps SDK by selecting **Windows -> Extensions**, and then select **Bing Maps for C#, C++ and Visual Basic** and **Microsoft Visual C++ 2013 Runtime Package**.



Check the references to the Bing Maps SDK in the **Microsoft.Maps.SpatialToolbox.Win8** project. It is possible that you may have a different version of Bing Maps. If this is the case, simply remove the Bing Maps references from this library and re-add the reference to the version of Bing Maps you have installed.

Next set the **Active solution platform** in Visual Studio by right clicking on the Solution folder and select **Properties**. Then go to **Configuration Properties -> Configuration**. Find your project and the **Microsoft.Maps.SpatialToolbox.Win8** project under the **Platform** column and set the target platform to x86 and press OK.



Open the **MainPage.xaml** file in the **SpatialDataViewer** project. In here we will add a button into the bottom app bar that when clicked, will open a **MenuFlyout** with a list of the different file format to import. In the main **Grid** of this page we will add a **Map** that has two **MapLayers**, one for pushpins and the other for an infobox. The infobox will consist of a **TextBox** where the **ShapeMetadata Title** will be displayed, and a **WebView** for the **Description** property. The reason for using a **WebView** is that in a lot of XML files it is common to include HTML in the description. By passing this into a **WebView** it will render the HTML for us. Update the **MainPage.xaml** file with the following XAML.

```
<Page
    x:Class="SpatialDataViewer.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:SpatialDataViewer"
```

```

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:m="using:Bing.Maps">

<Page.BottomAppBar>
    <AppBar>
        <AppBarButton Label="Import Data">
            <AppBarButton.Icon>
                <FontIcon Glyph="⌕" />
            </AppBarButton.Icon>
            <AppBarButton.Flyout>
                <MenuFlyout>
                    <MenuFlyoutItem Text="Well Known Text" Tapped="ImportData_Tapped" Tag="wkt"/>
                    <MenuFlyoutItem Text="Shapefile" Tapped="ImportData_Tapped" Tag="shp"/>
                    <MenuFlyoutItem Text="GeoRSS" Tapped="ImportData_Tapped" Tag="georss"/>
                    <MenuFlyoutItem Text="GPX" Tapped="ImportData_Tapped" Tag="gpx"/>
                    <MenuFlyoutItem Text="KML" Tapped="ImportData_Tapped" Tag="kml"/>
                    <MenuFlyoutItem Text="GeoJSON" Tapped="ImportData_Tapped" Tag="geojson"/>
                </MenuFlyout>
            </AppBarButton.Flyout>
        </AppBarButton>
    </AppBar>
</Page.BottomAppBar>

<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <m:Map Name="MyMap" Credentials="YOUR_BING_MAPS_KEY">
        <m:Map.Children>
            <m:MapLayer Name="PinLayer"/>
            <m:MapLayer Name="InfoboxLayer" Visibility="Collapsed">
                <Grid x:Name="Infobox" Width="350">
                    <Border Background="Black" Opacity="0.8" BorderBrush="White"
                        BorderThickness="2" CornerRadius="5"/>

                    <Grid Margin="5">
                        <StackPanel HorizontalAlignment="Left" Margin="10">
                            <TextBlock Name="InfoboxTitle" FontSize="18" Width="290"
                                HorizontalAlignment="Left" TextWrapping="Wrap" />

                            <WebView Name="InfoboxDescription" Width="330" Height="100"
                                Margin="0,10,0,0"/>
                        </StackPanel>

                        <Button Content="X" Tapped="CloseInfobox_Tapped"
                            HorizontalAlignment="Right" VerticalAlignment="Top"/>
                    </Grid>
                </Grid>
            </m:MapLayer>
        </m:Map.Children>
    </m:Map>
</Grid>
</Page>

```

In the **MainPage.xaml.cs** file we will add a **MapShapeLayer** to the map when the page is loading. This will be used for rendering polylines and polygons on the map. We will also need to add all the button event handlers, a common method for clearing the map, and create a default style to use for rendering shapes on the map. Add the following code to the **MainPage.xaml.cs** file.

C#

```

using Bing.Maps;
using Microsoft.Maps.SpatialToolbox;
using Microsoft.Maps.SpatialToolbox.Bing;
using Microsoft.Maps.SpatialToolbox.IO;
using System;
using System.IO;
using Windows.Storage.Pickers;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Input;

namespace SpatialDataViewer
{
    public sealed partial class MainPage : Page
    {
        private MapShapeLayer PolyLayer;

        private ShapeStyle DefaultStyle = new ShapeStyle()
        {
            FillColor = new StyleColor() { A = 150, B = 255 },
            StrokeColor = new StyleColor() { A = 150 },
            StrokeThickness = 4
        };

        public MainPage()
        {
            this.InitializeComponent();

            PolyLayer = new MapShapeLayer();
            MyMap.ShapeLayers.Add(PolyLayer);

            private async void ImportData_Tapped(object sender, TappedRoutedEventArgs e)
            {
            }

            private void CloseInfobox_Tapped(object sender, TappedRoutedEventArgs e)
            {
                InfoboxLayer.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
            }

            private void ClearMap()
            {
                PinLayer.Children.Clear();
                PolyLayer.Shapes.Clear();
                InfoboxLayer.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
            }
        }
    }
}

```

Visual Basic

```

Imports Bing.Maps
Imports Microsoft.Maps.SpatialToolbox
Imports Microsoft.Maps.SpatialToolbox.Bing
Imports Microsoft.Maps.SpatialToolbox.IO
Imports Windows.Storage.Pickers

```



```

Public NotInheritable Class MainPage
    Inherits Page

    Private PolyLayer As MapShapeLayer
    Private DefaultStyle As ShapeStyle

    Public Sub New()
        InitializeComponent()

        'Add a MapShapeLayer to map for polylines and polygons.
        PolyLayer = New MapShapeLayer()
        MyMap.ShapeLayers.Add(PolyLayer)

        'Define Default style
        DefaultStyle = New ShapeStyle()

        Dim fill = New StyleColor()
        fill.A = 150
        fill.B = 255

        DefaultStyle.FillColor = fill

        Dim stroke = New StyleColor()
        stroke.A = 150

        DefaultStyle.StrokeColor = stroke
        DefaultStyle.StrokeThickness = 4
    End Sub

    Private Async Sub ImportData_Tapped(sender As Object, e As TappedRoutedEventArgs)
    End Sub

    Private Sub CloseInfobox_Tapped(sender As Object, e As TappedRoutedEventArgs)
        InfoboxLayer.Visibility = Windows.UI.Xaml.Visibility.Collapsed
    End Sub

    Private Sub ClearMap()
        PinLayer.Children.Clear()
        PolyLayer.Shapes.Clear()
        InfoboxLayer.Visibility = Windows.UI.Xaml.Visibility.Collapsed
    End Sub
End Class

```

Next we will add the logic for loading in the spatial data. To do this we will need to first determine the type of spatial data file to read, and then create an instance of the appropriate spatial file reader from the spatial toolbox library. Once this is done we will then need to render the data on the map. Update the **ImportData_Tapped** event handler with the following code.

C#

```

private async void ImportData_Tapped(object sender, TappedRoutedEventArgs e)
{
    try
    {
        var selectedItem = sender as Windows.UI.Xaml.Controls.MenuFlyoutItem;

        BaseFeed reader = null;
        string[] fileTypes = null;
    }
}

```

```

//Check to see if the Tag property of the selected item has a string that is a
spatial file type.
if (selectedItem != null)
{
    switch (selectedItem.Tag.ToString())
    {
        case "wkt":
            reader = new WellKnownText();
            fileTypes = new string[] { ".txt" };
            break;
        case "shp":
            reader = new ShapefileReader();
            fileTypes = new string[] { ".shp" };
            break;
        case "gpx":
            reader = new GpxFeed();
            fileTypes = new string[] { ".xml", ".gpx" };
            break;
        case "georss":
            reader = new GeoRssFeed();
            fileTypes = new string[] { ".xml", ".rss" };
            break;
        case "kml":
            reader = new KmlFeed();
            fileTypes = new string[] { ".xml", ".kml", ".kmz" };
            break;
        case "geojson":
            reader = new GeoJsonFeed();
            fileTypes = new string[] { ".js", ".json" };
            break;
        default:
            break;
    }
}

if (reader != null && fileTypes != null)
{
    ClearMap();

    //Create a FileOpenPicker to allow the user to select which file to import
    var openPicker = new FileOpenPicker()
    {
        ViewMode = PickerViewMode.List,
        SuggestedStartLocation = PickerLocationId.Desktop
    };

    //Add the allowed file extensions to the FileTypeFilter
    foreach (var type in fileTypes)
    {
        openPicker.FileTypeFilter.Add(type);
    }

    //Get the selected file
    var file = await openPicker.PickSingleFileAsync();
    if (file != null)
    {
        using (var fileStream = await file.OpenStreamForReadAsync())
        {
            //Read the spatial data file
            var data = await reader.ReadAsync(fileStream);

```



```

        fileTypes = New String() {".xml", ".kml", ".kmz"}
        Exit Select
    Case "geojson"
        reader = New GeoJsonFeed()
        fileTypes = New String() {".js", ".json"}
        Exit Select
    End Select
End If

If (reader IsNot Nothing AndAlso fileTypes IsNot Nothing) Then
    ClearMap()

    'Create a FileOpenPicker to allow the user to select which file to import
    Dim openPicker = New FileOpenPicker()
    openPicker.ViewMode = PickerViewMode.List
    openPicker.SuggestedStartLocation = PickerLocationId.Desktop

    'Add the allowed file extensions to the FileTypeFilter
    For Each type As String In fileTypes
        openPicker.FileTypeFilter.Add(type)
    Next

    'Get the selected file
    Dim file = Await openPicker.PickSingleFileAsync()
    If (file IsNot Nothing) Then
        Using fileStream As Stream = Await file.OpenStreamForReadAsync()
            'Read the spatial data file
            Dim data = Await reader.ReadAsync(fileStream)

            If (String.IsNullOrEmpty(data.Error)) Then
                'Load the spatial data set into the map
                MapTools.LoadGeometries(data, PinLayer, PolyLayer, DefaultStyle,
AddressOf DisplayInfobox)

                'If the data set has a bounding box defined, use it to set the map
view.

                If (data.BoundingBox IsNot Nothing) Then
                    MyMap.SetView(data.BoundingBox.ToBMGeometry())
                End If
            Else
                'If there is an error message, display it to the user.
                Dim msg = New Windows.UI.Popups.MessageDialog(data.Error)
                Await msg.ShowAsync()
            End If
        End Using
    End If
End If
Catch
End Try
End Sub

```

The last bit of functionality to add is a method for displaying the infobox. When a user taps on a pushpin or shape we will be able to retrieve the **ShapeMetadata** from the **Tag** property. This information can then be used to populate the **InfoboxTitle** and **InfoboxDescription** properties. Open the **MainPage.xaml.cs** file and add the following method.

C#

```

private void DisplayInfobox(object sender, TappedRoutedEventArgs e)
{
    ShapeMetadata tagMetadata = null;

```

```

Location anchor = null;

if(sender is FrameworkElement){
    var pin = sender as Windows.UI.Xaml.FrameworkElement;
    anchor = MapLayer.GetPosition(pin);

    //Get the stored metadata from the Tag property
    tagMetadata = pin.Tag as ShapeMetadata;
}else if (sender is MapShape){
    var shape = sender as MapShape;
    tagMetadata = shape.GetValue(MapShapeExt.TagProperty) as ShapeMetadata;
    anchor = shape.ToGeometry().Envelope().Center.ToBMGeometry();
}

if (anchor != null && tagMetadata != null)
{
    bool hasContent = false;

    if (!string.IsNullOrEmpty(tagMetadata.Title))
    {
        //Set the Infobox title and description using the Metadata information
        InfoboxTitle.Text = tagMetadata.Title;
        hasContent = true;
    }

    if (!string.IsNullOrEmpty(tagMetadata.Description))
    {
        //Since the description value is being passed to a WebView, use the
        NavigateToString method to render the HTML.
        InfoboxDescription.NavigateToString(tagMetadata.Description);
        InfoboxDescription.Visibility = Windows.UI.Xaml.Visibility.Visible;
        hasContent = true;
    }
    else
    {
        InfoboxDescription.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
    }

    if (hasContent)
    {
        //Display the infobox
        InfoboxLayer.Visibility = Windows.UI.Xaml.Visibility.Visible;
    }
    else
    {
        InfoboxLayer.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
    }

    //Set the position of the infobox
    MapLayer.SetPosition(Infobox, anchor);
}
}

```

Visual Basic

```

Private Sub DisplayInfobox(sender As Object, e As TappedRoutedEventArgs)
    Dim tagMetadata As ShapeMetadata = Nothing
    Dim anchor As Location = Nothing

    If TypeOf sender Is FrameworkElement Then

```

```

Dim pin = TryCast(sender, Windows.UI.Xaml.FrameworkElement)
anchor = MapLayer.GetPosition(pin)

'Get the stored metadata from the Tag property
tagMetadata = TryCast(pin.Tag, ShapeMetadata)
ElseIf TypeOf sender Is MapShape Then
    Dim shape = TryCast(sender, MapShape)
    tagMetadata = TryCast(shape.GetValue(MapShapeExt.TagProperty), ShapeMetadata)
    anchor = shape.ToGeometry().Envelope().Center.ToBMGeometry()
End If

If anchor IsNot Nothing AndAlso tagMetadata IsNot Nothing Then
    Dim hasContent As Boolean = False

    If Not String.IsNullOrEmpty(tagMetadata.Title) Then
        'Set the Infobox title and description using the Metadata information
        InfoboxTitle.Text = tagMetadata.Title
        hasContent = True
    End If

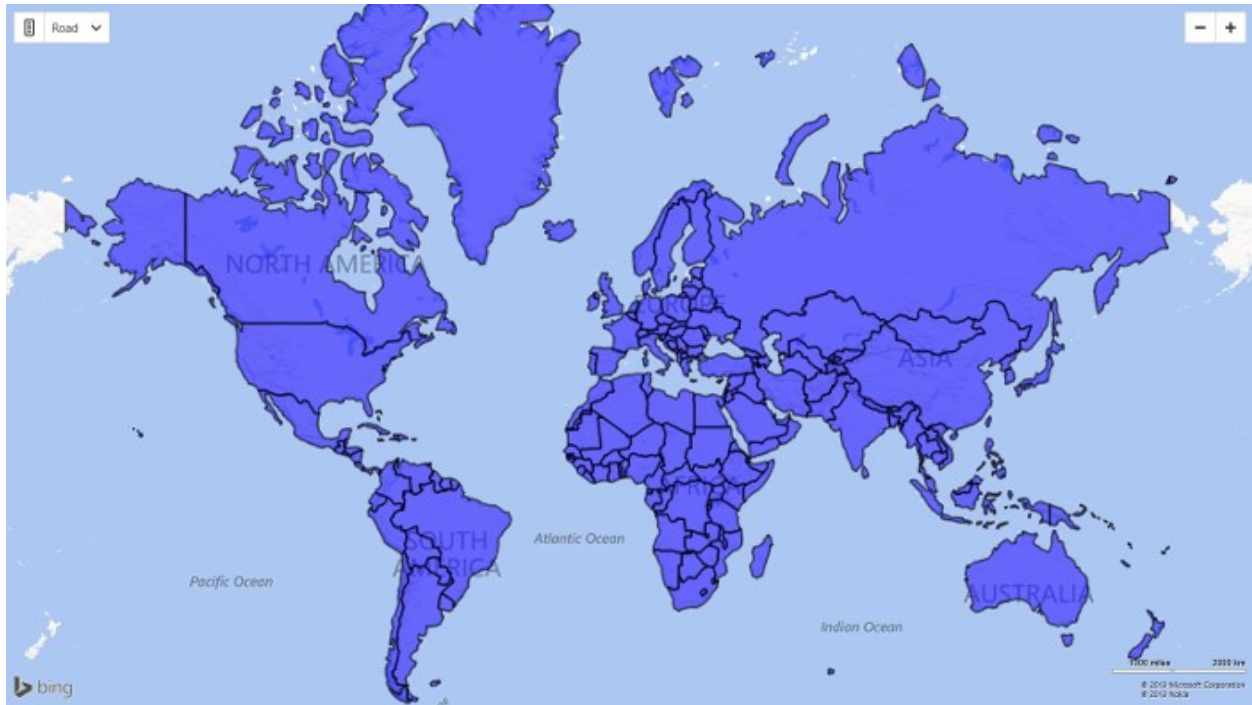
    If Not String.IsNullOrEmpty(tagMetadata.Description) Then
        'Since the description value is being passed to a WebView, use the
        NavigateToString method to render the HTML.
        InfoboxDescription.NavigateToString(tagMetadata.Description)
        InfoboxDescription.Visibility = Windows.UI.Xaml.Visibility.Visible
        hasContent = True
    Else
        InfoboxDescription.Visibility = Windows.UI.Xaml.Visibility.Collapsed
    End If

    If hasContent Then
        'Display the infobox
        InfoboxLayer.Visibility = Windows.UI.Xaml.Visibility.Visible
    Else
        InfoboxLayer.Visibility = Windows.UI.Xaml.Visibility.Collapsed
    End If

    'Set the position of the infobox
    MapLayer.SetPosition(Infobox, anchor)
End If
End Sub

```

Now run your application by pressing F5 or the Debug button. Right click or swipe in from the top or bottom to bring up the bottom app bar. Press the import button and select a spatial file type to import into the map. The file open picker will appear, navigate to a spatial file you want to import and select it. Here is a screenshot of country boundaries that were imported from a GeoJSON file.



Windows Runtime Components

The spatial toolbox library is great if you are working with C# or Visual Basic, but what if you are using JavaScript. Wouldn't it be great if you could use some of these tools in JavaScript too? Well we are in luck, one of the great features in Windows 8 is the ability to mix .NET and C++ libraries with JavaScript applications. This can be done by using what is called a Windows Runtime Component.

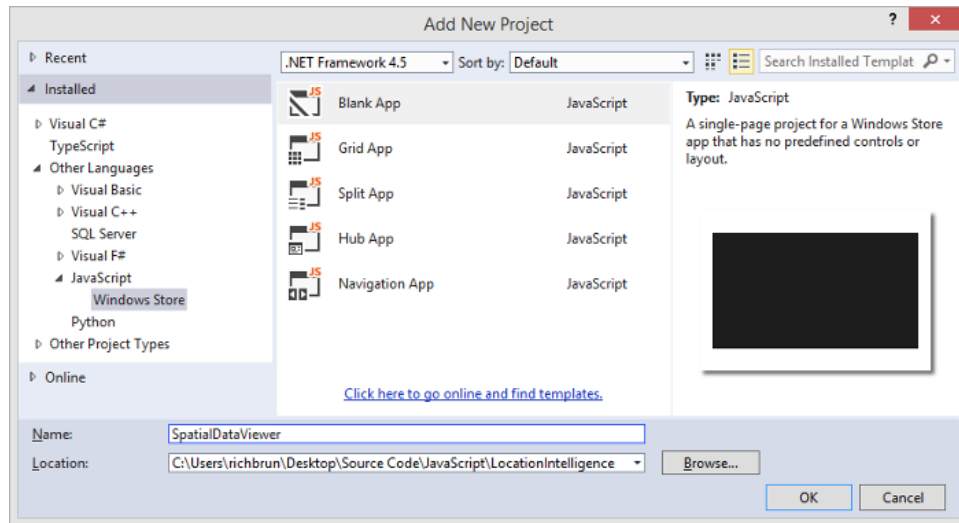
Windows Runtime Components are also a great way to offload computational intensive operations to other languages that are able to do the same operation with better performance. For instance C++ is capable of performing computational tasks much faster than C#, Visual Basic, or JavaScript. If you have a part of your application that requires a lot of calculations then it might make sense to create a Windows Runtime Component with this task written in C++. This is done in the Bing Maps Trip Optimizer code sample available on MSDN. You can find an excellent write up on creating this code sample here: <http://bit.ly/1gD9v10>

Windows Runtime Components are a very useful way of sharing code. However, a Windows Runtime Component is not a standard DLL or assembly and there are a number requirements you must follow such as only using allowed types and making all public classes sealed. Documentation on how to create Windows Runtime Components can be found here: <http://bit.ly/1bu7U90>

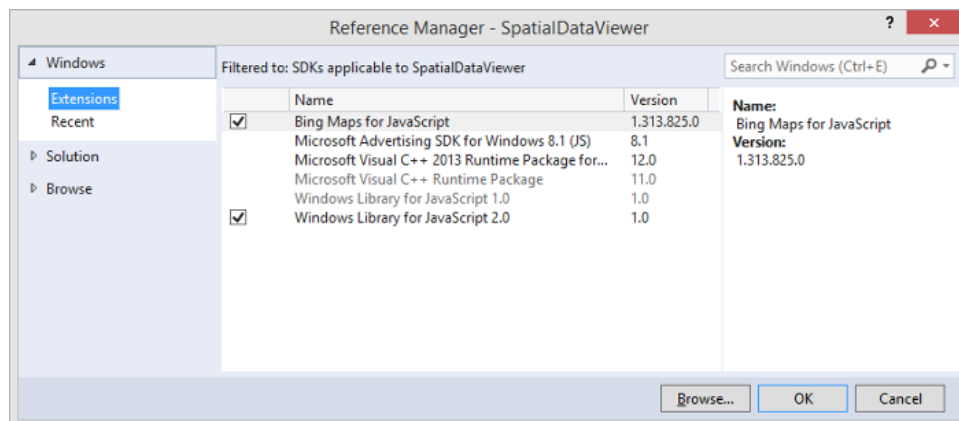
Creating a Spatial Data Viewer in JavaScript

As we saw in the previous section a Windows Runtime Component can be used to connect the spatial toolbox library which is written in C# to a JavaScript application. In fact the spatial toolbox project on CodePlex includes a simple Windows Runtime Component that makes it easy to read spatial data files from JavaScript. In this section we are going to see how to connect this Windows Runtime Component to a JavaScript Windows Store app using Bing Maps.

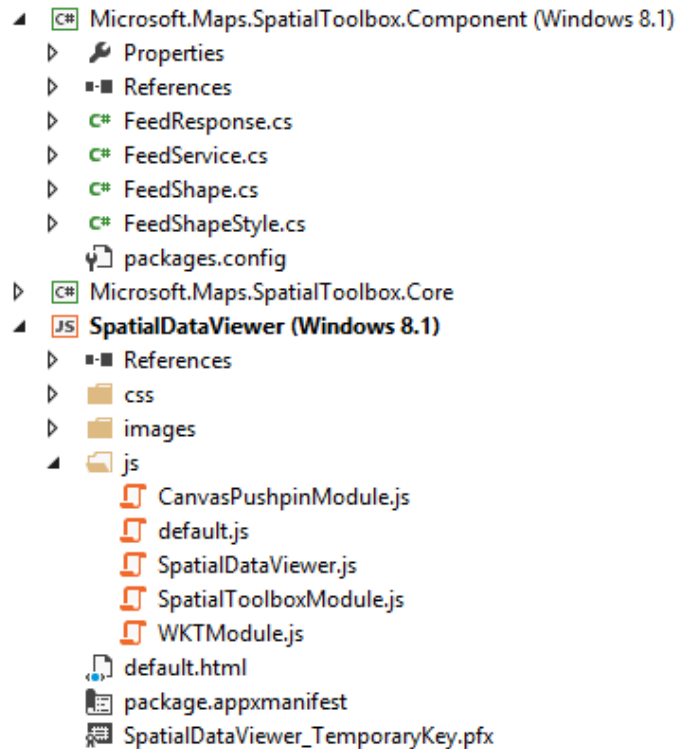
To get started open up Visual Studios and create a new JavaScript project. Select the **Blank App** template and call the application **SpatialDataViewer** and press **OK**.



Next get a copy of the spatial toolbox library. Either copy it from the code samples for this book or download it from the CodePlex site (<https://mapstoolbox.codeplex.com/>). Navigate to the folder where the **SpatialDataViewer** project is located and add a copy of the **Microsoft.Maps.SpatialToolbox.Core** and **Microsoft.Maps.SpatialToolbox.Component** projects to that folder. Add these projects to the solution in Visual Studios by righting click on the solution folder and selecting **Add -> Existing Project**. Next add a references to these projects in the **SpatialDataViewer** by right click on the **References** folder and press **Add Reference**. Select **Solution -> Projects**, and then select the **Microsoft.Maps.SpatialToolbox.Core** and **Microsoft.Maps.SpatialToolbox.Component**. While you are here add a reference to the Bing Maps SDK by selecting **Windows -> Extensions**, and then select **Bing Maps for JavaScript**.



Next download the Canvas Pushpin and the Well Known Text modules from the Bing Maps V7 Modules project on CodePlex (<http://bingmapsv7modules.codeplex.com/>) and copy the JavaScript files into the **js** folder. The Canvas Pushpin module uses the HTML5 canvas to create pushpins. This allows us to create complex pushpins through code without the need for an image based icon. From the spatial toolbox project locate the **SpatialToolboxModule.js** file in the code samples and copy this file into the **js** folder. This module makes it easy to add the data in the response from the spatial toolbox component to the Bing Maps control as a layer. Next create a new JavaScript file by right clicking on the **js** folder and selecting **Add -> New item**. Call the file **SpatialDataViewer.js**. We will use this file to put all the application logic into. At this point your Solution folder will look something like this:



Open the **default.html** file in the **SpatialDataViewer** project. In here we will add references to the Bing Maps SDK and all the JavaScript files for the modules and application logic. In the body of the page we will add a **div** for the map and a button into the bottom app bar that when clicked, will open a **MenuFlyout** with a list of the different file format to import. Update the **default.html** file with the following HTML.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>SpatialDataViewer</title>

  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
  <script src="//Microsoft.WinJS.2.0/js/base.js"></script>
  <script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

  <!-- SpatialDataViewer references -->
  <link href="/css/default.css" rel="stylesheet" />
  <script src="/js/default.js"></script>

  <!-- Bing Map Control references -->
  <script type="text/javascript" src="ms-appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>
  <script type="text/javascript" src="ms-appx:///Bing.Maps.JavaScript/js/veapiModules.js"></script>

  <!-- Load the WKT Module -->
  <script>Microsoft.Maps.registerModule('WKTModule');</script>
  <script type="text/javascript" src="/js/WKTModule.js"></script>

  <!-- Load the Canvas Puushpin Module -->
  <script>Microsoft.Maps.registerModule('CanvasPushpinModule');</script>
  <script type="text/javascript" src="/js/CanvasPushpinModule.js"></script>
```

```

<!-- Load the Spatial Toolbox Module -->
<script>Microsoft.Maps.registerModule('SpatialToolboxModule');

```

Now open the **SpatialDataViewer.js** file. In this file add the logic for loading the map and the modules. When all the modules are loaded create an instance of the **MapTools** class from the spatial toolbox module. This will make it easy to add the response data from the spatial toolbox module to the map. Add two layers to the map, one for shapes (polygons and polylines) and one for canvas pushpins. Add click events for all of the spatial file type import buttons. When a click occurs, a file picker will open and allow the user to select the file they want to import. Take the following code that does all this and add it to the **SpatialDataViewer.js** file.

```

(function () {
  var map, mapTools, shapeLayer;

  function initialize() {
    Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: GetMap });

    // Initialize WinJS controls.
    WinJS.UI.processAll();
  }

```

```

document.getElementById("WKTBtn").addEventListener("click", function (e) {
    ImportData_Tapped("wkt", [".txt"]);
}, true);

document.getElementById("ShapefileBtn").addEventListener("click", function (e) {
    ImportData_Tapped("shp", [".shp"]);
}, true);

document.getElementById("GeoRSSBtn").addEventListener("click", function (e) {
    ImportData_Tapped("georss", [".xml", ".rss"]);
}, true);

document.getElementById("GpxBtn").addEventListener("click", function (e) {
    ImportData_Tapped("gpx", [".xml", ".gpx"]);
}, true);

document.getElementById("KmlBtn").addEventListener("click", function (e) {
    ImportData_Tapped("kml", [".xml", ".kml", ".kmz"]);
}, true);

document.getElementById("GeoJSONBtn").addEventListener("click", function (e) {
    ImportData_Tapped("geojson", [".js", ".json"]);
}, true);
}

function GetMap() {
    map = new Microsoft.Maps.Map(document.getElementById("MyMap"), {
        credentials: "YOUR_BING_MAPS_KEY"
    });

    //Add a layer for storing shape data in
    shapelayer = new Microsoft.Maps.EntityCollection();
    map.entities.push(shapelayer);

    //Load Well Known Text and Canvas Pushpin Modules
    Microsoft.Maps.loadModule('WKTModule');
    Microsoft.Maps.loadModule('CanvasPushpinModule', {
        callback: function () {
            //Create Canvas Entity Collection
            var pinLayer = new CanvasLayer(map);
            map.entities.push(pinLayer);

            //Load the Spatial Toolbox Module
            Microsoft.Maps.loadModule('SpatialToolboxModule', {
                callback: function () {
                    mapTools = new MapTools(map, pinLayer, shapelayer)
                }
            });
        }
    });
}

function ImportData_Tapped(feedType, fileTypes) {
    //Clear the data from the map
    mapTools.ClearLayers();

    //Create a file picker
    var openPicker = new Windows.Storage.Pickers.FileOpenPicker();
    openPicker.viewMode = Windows.Storage.Pickers.PickerViewMode.list;

```

```

    openPicker.suggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.desktop;

    var list = new WinJS.Binding.List(fileTypes);
    list.forEach(function(value, index, array){
        openPicker.fileTypeFilter.push(value);
    });

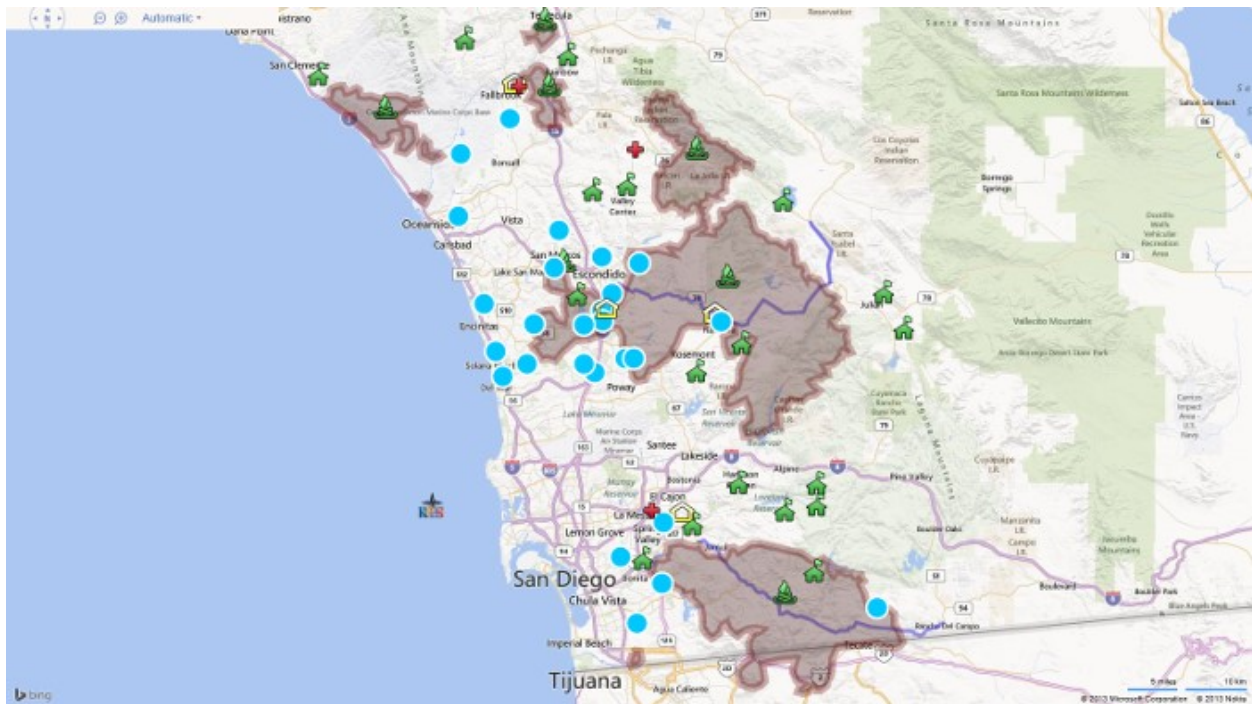
    //Open a file
    openPicker.pickSingleFileAsync().done(
        function (file) {
            if (file != null) {
                //Read the file and pass the random access stream to the SpatialToolbox
                component to parse the file.
                file.openReadAsync().then(function (stream) {

Microsoft.Maps.SpatialToolbox.Component.FeedService.readFeedStreamAsync(stream,
feedType).then(function (data) {
                    //Load the spatial data into the map.
                    mapTools.LoadFeedShapes(data);
                });
            }
        });
    });
}

document.addEventListener("DOMContentLoaded", initialize, false);
})();

```

Now run your application by pressing F5 or the Debug button. Right click or swipe in from the top or bottom to bring up the bottom app bar. Press the import button and select a spatial file type to import into the map. The file open picker will appear, navigate to and select the spatial file you want to import. Here is a screenshot of an imported KML file containing information about the 2007 California fires.

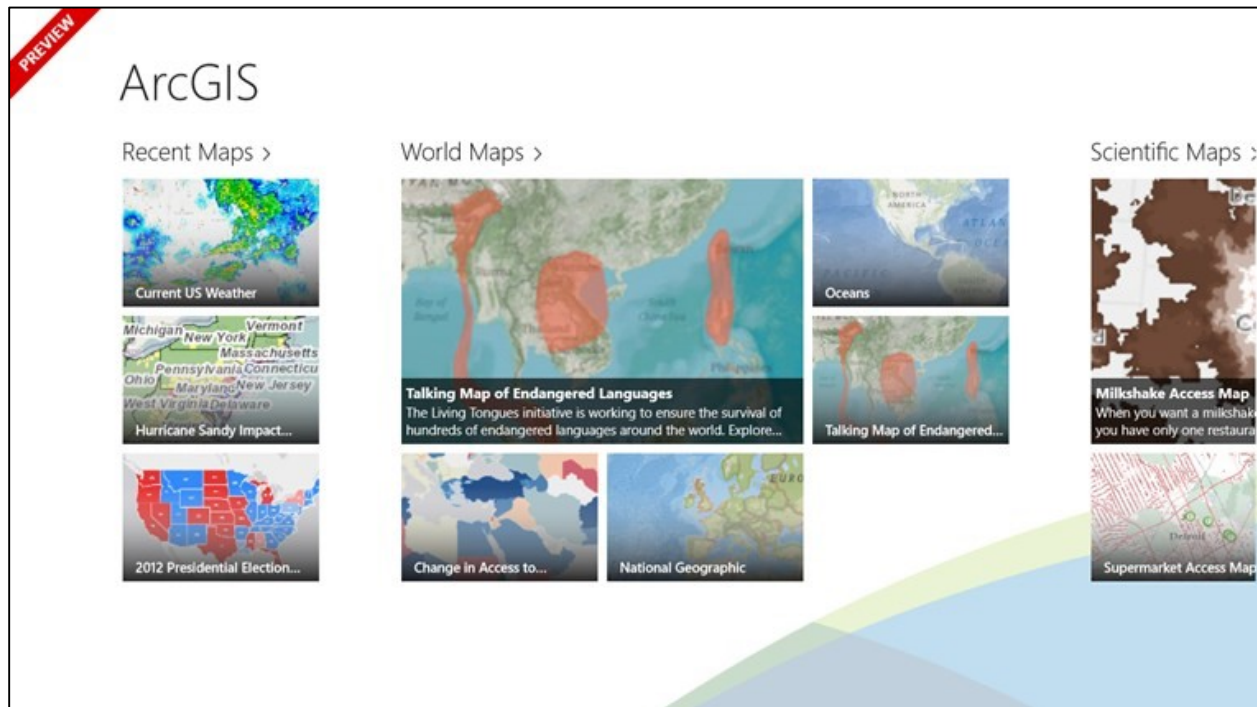


Real World Example: ESRI ArcGIS Runtime SDK for .NET (Beta)

In addition to Bing Maps and the spatial toolbox library covered in this chapter there are other great tools for creating spatial Windows Store applications. One of these great tools is the ESRI ArcGIS Runtime SDK for .NET, which is currently in beta. In addition to being able to use it in Windows Store apps this SDK can also be used in Windows Phone and WPF applications as well. You can find this toolkit on ESRI's ArcGIS website here:

<https://developers.arcgis.com/en/windows-store>

ESRI has created a great Windows Store app called ArcGIS that showcases many of the great feature in this SDK. With this app you can explore a gallery of compelling maps that showcase relevant demographic, commercial, and environmental topics. Use the app to view, navigate, and learn more about these maps with your fingertips.



You can download this app from the Windows Store here: <http://bit.ly/1edlai4>

Connecting Spatial Data to your app

So far in this chapter we have seen a number of different file types that are commonly used to store spatial data. We also saw how to import these files from the users file system into a Windows Store application. This is great if you want to provide the user with the ability to import their own data into the map however chances are you will want to power the application with your own spatial data. In this case there are number of different options for hosting the data. The two main options are embedding the file directly into the application or hosting it online.

Embedding the spatial data file in the app may be a good option but there are some things to consider before going this route. The only option you have to update the data is to create an update for your app in the Windows Store. This could take a couple of weeks to get the new data to your users depending on how long it takes for your app to pass the Windows Store review process. If your data doesn't change that often then this may be an acceptable update process. Another thing to take into consideration is the size of the data. Since the user will be installing the app to their device any embedded content will also be stored on the user's device, thus the larger the embedded file is the

more space your app will take up on the user's device. This could be an issue on devices that have a small amount of storage. If your spatial data file is larger than a few megabytes then consider hosting it online instead of embedding it in the app.

When hosting data online it might be tempting to simply upload your file somewhere and import the whole file using the URL into your app, but here are a couple of tips to make your application better.

- Only load the data you need when you need it.
- Minimize the amount of data as much as possible.
- Compress the data.

Many developers prefer to load all the data into the application all at once because they see it as being a quick and easy solution. Unfortunately, more often than not this approach usually has a number of side effects such as slow loading times, poor user experience or performance issues. In some cases the data sets may simply be too big to even load all at once. In general it is best to only load the data that is needed, when it is needed. Say for example you have a data set that spans across the globe but the user is zoomed into their city. They only need the data for their city, not the rest of the world so why not limit the data that is loaded to the area the user is zoomed into.

If the data is being hosted remotely on a server the size of the data being sent to the application can have an impact on the time it takes to download the data. Many people prefer to use XML as it is well structured and easy to read, but XML is often bloated in size. JSON is a much better way to transfer data as it tends to be 20%-30% smaller. You can further reduce the size of the data set the user has to download by compressing the coordinates using the **PointCompression** algorithm that is available in the spatial toolbox library. This algorithm turns an array of coordinates into a 64bit encoded string. This significantly reduces the overall size of the data. This approach comes with a trade off in that you have to decode the string in the application, however in most cases this approach tends to be faster than simply downloading the larger data set. This approach is also great for situations where you want to limit the amount of bandwidth your application uses, such as for mobile application. You can also further minimize the size of the data by enabling GZip on the server. If GZip is not an option then consider hosting a zipped version of your data online and unzipping it in your application.

Bing Spatial Data Services

As we saw in chapter 6 the Bing Spatial Data Services is an excellent option for hosting your spatial data. It provides an easy way to upload and expose your data as a REST service complete with several common spatial queries such as find nearby, find by bounding box, find along route, find within a geometry and find by property.

OneDrive

OneDrive (formerly known as SkyDrive) has been around for a while and is an excellent place to store your files in the cloud, however many developers don't realize that it is also a great location for your app to store files too. That's right, it is possible to integrate OneDrive into your app.

Why might you want to do this? Take this scenario as an example. You are creating a simply Hiking Trail app that allows you to use the GPS to track where you have been hiking. Now let's say the user wants to view the Hiking Trail route they created on another device or they want to share it with someone else. In this case storing the data in a GPX file would make sense as that's one of the most common file types used by GPS devices. By allowing the user to upload this GPX file to OneDrive they can easily share it with anyone they want. They can also easily access it from any device that has access to OneDrive too.

Using OneDrive to share files between different apps is also a great idea. Let's say that you create a Windows Phone app for capturing and creating the hiking trail information. This would make more sense as the user would be more likely to carry their phone than their computer when on a hike. A mobile device is great for capturing the data when on the go but when you want to view the information the smaller screen size could be a bit of a pain. A companion Windows Store app that loads the GPX files from a user's OneDrive would make a lot of sense in this case. Not only

does this make it easier for the user but is also creates a seamless experience across multiple devices which is likely to get the user using more of your apps.

Documentation on the OneDrive API can be found here: <http://bit.ly/17Y0XAI>

You can also find documentation on how to use the OneDrive API inside of a Windows Store app here: <http://bit.ly/1aY5tz3>

Storing data locally with SQLite

SQLite is a popular open source database system that can easily be embedded into a Windows Store application. This is very similar to SQL Express which is a local database that is self-contained, server-less, requires zero-configuration, and is a transactional SQL database. Using SQLite to manage your data locally is much more efficient than using flat files such as XML files. In addition to being more efficient it is often much smaller and also easier to use, especially if you want to filter the data and run queries against it. SQLite is also supported on multiple platforms which means you will be able to reuse your database in different applications. There are a lot of different scenarios where you might find using SQLite useful. Here are a few examples of where you might use them.

- Embed small data sets in Windows Store apps to save you the work of hosting the data online.
- Save application states locally so that the next time the user opens the application it restores their last session so they can continue from where they left off.
- Allow your application to capture information locally when offline and let it sync the data up later with a web service. This is useful if your end users will be capturing data out in the field and will need to add it to a centralized system later.

Johannes Kebeck from the Bing Maps team wrote a great blog post on how to use SQLite in a spatial Windows Store app (<http://binged.it/19RqH12>). In this blog post he created a SQLite database from an ESRI shapefile and stored the spatial data as Well Known Text. I personally prefer to manage my databases inside of a locally installed version of SQL Server 2012 and then convert them to SQLite databases as needed using this free tool: <http://bit.ly/1icDfC4>

Tip: Rather than storing the spatial data as Well Known Text consider storing a Well Known Binary version as a binary blob. The reason for this is that Well Known Binary takes less time to parse than Well Known Text and the SQLite database will also be smaller.

Using SQLite inside your Window Store app requires installing the SQLite package into your app. To do this you will need to the NuGet package from Visual Studios: <http://bit.ly/1dP6Qfe>. Once installed it's fairly easy to use as queries against a SQLite database can be done using LINQ.

There is a spatial version of SQLite called SpatialLite which provides a number of spatial queries, however at the time of writing this book there has not been a Windows Store app version made available.

Static Tile Layers

In chapters 3 and 4 we saw that Bing Maps supports tile layers. Tile layers are an excellent way to visualize large data sets on a map with minimal impact on performance. A static tile layer is one in which the tiles are created ahead of time and are stored on a server just like any other image file. Here are some common items that are used to create static tile layers:

- Floor plans of buildings
- Historical maps
- High resolution imagery such as the weather imagery from NASA

If you have an image that you want to turn into a static tile layer you can use MapCruncher to chop it up. MapCruncher is a Microsoft Research project that makes it easy to cross reference an image with a set of locations on a map and then turn the image into a tile layer. You can download it from here: <http://bit.ly/1aAAkvn> and find a great

tutorial on how to use it here: <http://bit.ly/1i1HZu1>. You can also find a good blog post on using MapCruncher with the Bing Maps JavaScript control here: <http://bit.ly/1cOlaud>.

By default MapCruncher will generate a test application for you that is built using Bing Maps V6.3. You won't need this app, just the tiles, so simply locate the folder from the output directory that contains the map tiles and upload these to a server to host them.

Dynamic Tile Layers

Dynamic tile layers are a great way to display large data sets or data that changes frequently. Instead of creating the tiles ahead of time, a dynamic tile layer is a web service that generates the tiles on the fly from raw data. This type of application usually stores the raw spatial data in a database which is then connected to a web service. Logic in the web service then makes use of drawing tools such as **System.Drawing** in .NET on the server to create the map tiles. This approach isn't limited to vector spatial data, you can also use this to dynamically crop large images into tiles on the fly.

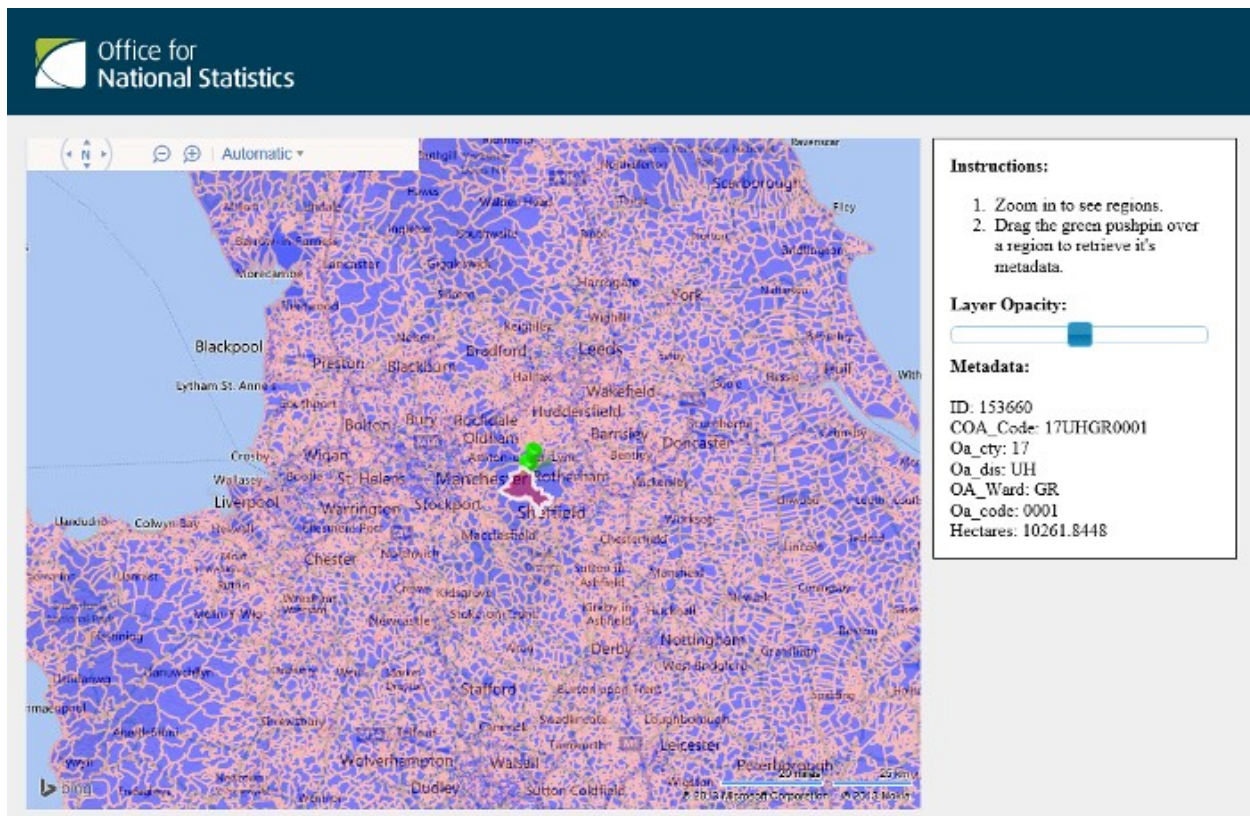
It's not overly difficult to create your own dynamic tile layer service. In fact I wrote a blog post on how to do this many years ago (<http://bit.ly/1a145ti>). The general process that a dynamic tile service goes through when a tile is requested is as follows:

1. Service receives a request for a tile by quad key.
2. Quad key boundaries calculated and used to retrieve data that intersects tile from database.
3. Custom business logic is used against data to determine how to render it.
4. The data is projected onto the tile and drawn on the image using drawing tools.
5. The service returns the tile as an image stream.

From a performance point of view you will want to ensure you implement caching of the tiles otherwise you may find your server unable to keep up with the requests. If your data is constantly changing and you don't want to use caching, even a short caching period of a few minutes can make a big difference in performance.

One reason why many developers have opted not to use dynamic tile layers in the past is that they believe that you can't interact with the data when it's on the map as a tile layer. This is partially correct, you won't be able to do too much with the tile layer itself however it is possible to interact with the data that powers the tile layer. Say for instance you want to fire an event when a user clicks on a shape on the map, but the shape is being displayed using a dynamic tile layer. What you can do is add the click event to the map and when this event fires, capture the location that was clicked and send it to a web service that is connected to your database. You can then use the spatial tools in the database to determine which shape the clicked location intersects with and then return the data back to the application so that the app can process the results. This approach works very well and is one I have used many times.

One such application I used this for was a demo for the UK Office of National Statistics. When I met with them they were hesitant to use an interactive map to display their data as they didn't think it would be possible to do this with any decent performance. I was provided with a couple of DVD's which contained high resolution census area boundaries. This data set was approximately 2GB in size and consisted of 175,000 polygons. Needless to say this is too much data to load directly into the map. The download process alone would take over an hour for most users. In this case the data is pretty static and it could have been turned into a static tile layer, but in the future I may want to change the color of the polygons based on data in which case the dynamic tile service approach would be much easier to update. Also, given the amount of area this data set covers a static tile layer would take up a lot more than 2GB of space. When it came to building the demo I opted to make use of Windows Azure and load all the data directly into SQL Azure. Once this was done I connected up a slightly modified version of the dynamic tile service I created in the blog post mentioned earlier to the database and added it to Bing Maps as a tile layer. To try something different I added a pushpin to the map that the user can drag. When they stop dragging it an event is fired to find metadata from the database for the area the pushpin is over top of. The performance of the application once it was completed, worked far better than I anticipated. You can try it out for yourself here <http://bit.ly/1e1rjji>.



Tip: One thing I've found when creating dynamic tile layers is that reducing the resolution of the data at higher zoom levels can really help with performance when drawing the data on a tile. If your data is stored in side of SQL Server 2008, 2012 or Azure you can use the **Reduce** method on the spatial data to reduce its resolution.

Microsoft Bing Maps MVP, Randy George, who works at OnTerra Systems in the US, wrote an excellent blog post on overlaying large data sets on Bing Maps which I highly recommend reading if you are working with a lot of data. You can find the post here: <http://bit.ly/1a7nLvF>

Custom Web Services

A custom web service is a great way to connect your application to data that's on a server. There are several different approaches on how to do this. One good approach is to create a WCF REST service that connects to a database using Entity Framework. The benefit of creating a REST based service is that you will be able to easily use it from any programming language. You can also have it return responses as XML or JSON very easily. JSON of course is the best to use for mobile applications as it is smaller than XML which means your user will have to download less data.

Entity Framework provides a set of tools that allow you to auto generate classes from database tables. It also provides you with easy-to-use functionality for connecting and querying your data using LINQ. Version 5 of Entity Framework and above support spatial data types. Entity Framework drastically reduces the amount of code you have to write when connecting to a database.

I have written two in depth articles on how to create spatial web services using WCF and Entity Framework 5 which you can find listed below.

- How to Create a Spatial Web Service That Connects a Database to Bing Maps Using EF5: <http://binged.it/1hW55zA>
- Advance Spatial Queries using Entity Framework 5: <http://binged.it/19MDfRe>

Windows Azure Mobile Service

Most great apps now a days are connected apps. The Windows Azure Mobile Services allow you to quickly create connected apps that can make use of cloud based storage, authenticate users, and send push notifications. With SDKs for Windows, Android, iOS, and HTML as well as a powerful and flexible REST API, Mobile Services lets you build connected applications for any platform and deliver a consistent experience across devices. Windows Azure Mobile Services makes configuring user authentication via Facebook, Google, Microsoft, or Twitter accounts as simple as a few clicks. Once users authenticate, not only can you personalize their experience, but also increase engagement and sharing. One of the great benefits of it being a cloud based service is that it has on-demand scaling allowing you to easily support more users as your app grows with just a few clicks of a button.

Windows Azure mobile services also have notification hubs which allow any app to broadcast push notifications to millions of devices with only a few lines of code. Notification hubs also enable pub/sub routing and tag-based multicast, which makes sending push notifications to a particular subset of users, such as those in a specific region, easy. Today, the Bing News app relies on notification hubs to deliver breaking news to millions of devices in minutes. Here are a couple of different scenarios where you might want to use Windows Azure Mobile Services:

- Sync application settings across multiple devices.
- Capture information from the user and store it in a database online. For instance, if you built a game you could easily sync the user's achievements and scores in a database using the Windows Azure Mobile Services. You could then use this database to provide ranking lists in your application so users can see how they are doing compared to other users.
- Have a service that notifies the user of changes to data in the service. In an asset tracking application it might be useful to allow your users to set up areas (geofences) that when the asset enters or leaves an area it triggers a notification that is displayed on the user's device.

As you may have already figured out the Windows Azure Mobile Services are part of Windows Azure. You can have up to 10 free Windows Azure Mobile Services running across 500 devices and 500,000 API calls per month. As your application grows in popularity you can upgrade to increase these limits.

Official documentation on Windows Azure Mobile Services can be found here: <http://bit.ly/1aVrry9>. Johannes Kebeck from the Bing Maps team has also written two great blog posts on using Windows Azure Mobile Services with Bing Maps.

- Windows Azure Mobile Services, Maps & More: <http://bit.ly/1cY1HHo>
- Traffic Notifications with Bing Maps & Windows Azure Mobile Services: <http://bit.ly/1crYYB8>

Real World Example: FloodAlerts

This app was created by a company called Shoothill who are Microsoft Partner in the UK and was built in cooperation with the UK Environment Agency. The FloodAlerts app takes real time flood data from the Environment Agency and transforms it in to a visible layer overlaid on a Bing Map, so you can assess your flood risk more accurately. When a flood alert is issued and affects your monitored location, a notification is sent to your Windows device. This gives you a direct link to the relevant alert(s) on the map, helping you assess the situation as it develops.

What's really impressive about this app is that the flood data updates every 15 minutes and can often be very large in size when there is a lot of flooding. Downloading all this data directly into the app would be slow and wouldn't work well. Instead this app makes use of Windows Azure to process the raw flood data into tiles that can be overlaid on top of the map. This approach makes it much easier to visualize massive amounts of data without any performance issues. This app is also able to use the spatial functionalities in SQL Azure to do complex spatial calculations such as determining if a location intersects with a flood area.

One benefit of using Windows Azure to manage the data is that it makes it much easier to target multiple platforms without the need to change the backend data processing functionality. As such not only is FloodAlerts available as a Window Store app but it is also available as a Facebook app and as a standalone webpage which you can find here: <http://bit.ly/19dAb0w>

This app is only available within the UK Windows Store. If you are interested in trying it out you can find it here: <http://bit.ly/1aOHNgv>



Chapter Summary

In this chapter you were introduced to several different types of spatial data and how to work with it in Windows Store apps. Here is a short summary of some key points from this chapter.

- The Open Geospatial Consortium (OGC) is a global organization that manages standards around geospatial data. The simple features standard consists of a set of **Geometry** classes used to represent different types of spatial data. These classes include; **Point**, **LineString**, **Polygon**, **MultiPoint**, **MultiLineString**, **MultiPolygon**, and **GeometryCollection**.
- Well Known Text (WKT) is an OGC standard that is used to represent spatial data in a textual format. Most OGC compliant systems such as SQL Server 2008, 2012 and Azure support Well Known Text. A WKT can only store the information for a single spatial object and this spatial data format is usually used as part of a larger file format or web service response.
- Well Known Binary (WKB) is another OGC standard for representing spatial objects in a uniform manner, in this case as binary. Like WKT this spatial data format is used to store a single spatial object.
- ESRI Shapefiles are a common binary file format developed by a company called ESRI. Many government agencies around the world provide free data in this format.
- KML stands for Keyhole Markup Language and is an XML file format that is also an OGC standard. It is used for storing spatial data along with styling and view information as well. As a result these files can grow in size quickly. This file format was popular for use in online mapping applications but has since been in decline as other, more efficient formats such as GeoJSON have become more mainstream.

- A GeoRSS file is an XML based web standard for embedding geospatial information inside of RSS or Atom feeds.
- GPX is yet another common XML format used for storing spatial data. This file format is commonly used by GPS systems and as such has become very popular.
- GeoJSON is a relatively new file format used for storing spatial data. This file format tends to use a lot less characters than XML equivalents thus resulting in a much smaller file size. This makes it ideal for transferring spatial data to web and mobile applications.
- Many JavaScript modules are available for importing spatial data into Bing Maps. The Bing Maps V7 Modules CodePlex project is a great source for these: <http://bingmaps7modules.codeplex.com/>
- The spatial toolbox library is an open source project on CodePlex that combines a number of tools together for working with spatial data. These consist of reader and writers for several spatial data formats, several common spatial calculations, and a number of tools that make it easy to integrate this data with Bing Maps in not only Windows Store apps but in Windows Phone and WPF apps as well. This library can be found on CodePlex here: <http://mapstoolbox.codeplex.com/>
- The Pushpin class in Bing Maps has a property called **Tag**. This property allows us to store any object in it. This is handy for linking a pushpin to its associated metadata. Unfortunately the **MapPolygon** and **MapPolyline** classes do not have this property. The **MapShapeExt** class provides a **DependencyProperty** that exposes a **Tag** property on the **MapPolygon** and **MapPolyline** classes. This **Tag** property can be used to store and retrieve an **object** from a **MapPolygon** or **MapPolyline** as shown in the following example.

C#

```
MapPolygon p = new MapPolygon();
p.SetValue(MapShapeExt.TagProperty, "My Metedata");
object tagValue = p.GetValue(MapShapeExt.TagProperty);
```

Visual Basic

```
Dim p = New MapPolygon()
p.SetValue(MapShapeExt.TagProperty, "My Metedata")
Dim tagValue = p.GetValue(MapShapeExt.TagProperty)
```

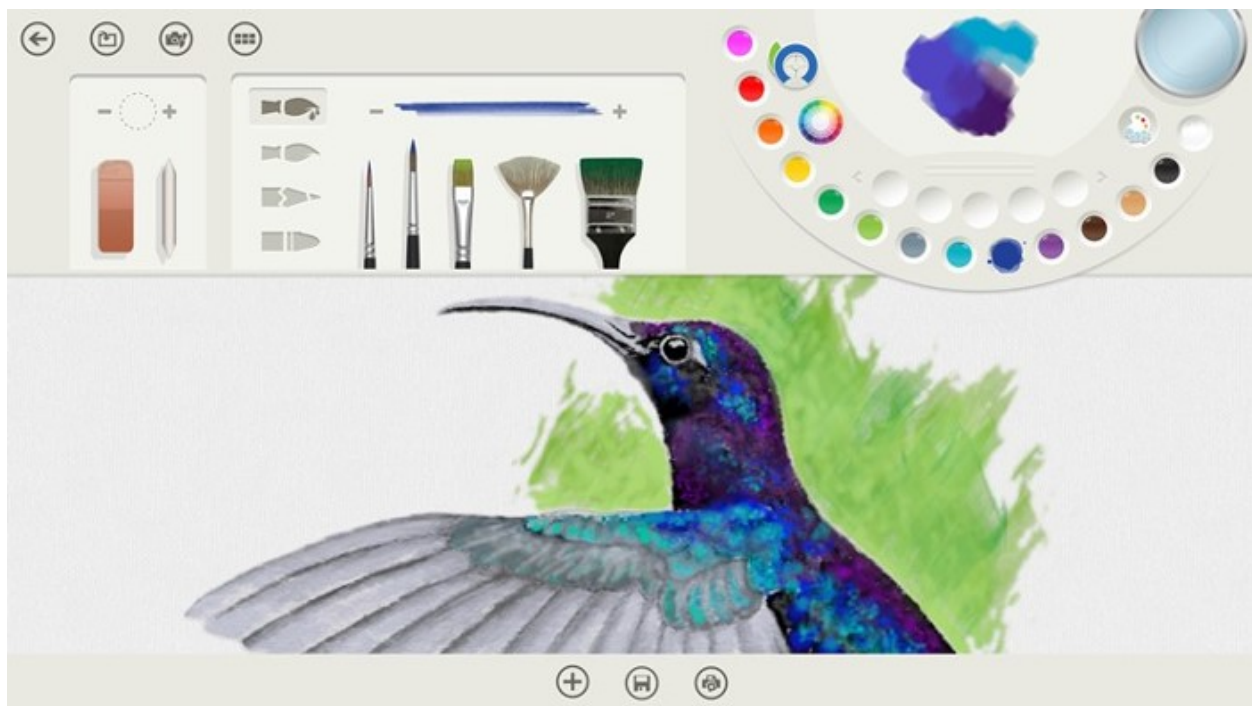
- Clustering is the process of grouping together nearby pushpins and representing them as a single location on the map. As the user zooms into the map the clusters will break apart to reveal the individual locations. This is a great way to improve the users experience when visualizing a lot of point based data on the map.
- The spatial toolbox project originated as a library for .NET Windows Store apps but can also be used by JavaScript Windows Store apps by using a Windows Runtime Component.
- Windows Runtime Components are a great way to offload computational intensive operations to other languages that are able to do the same operation with better performance. For instance C++ is capable of performing computational tasks much faster than C#, Visual Basic, or JavaScript.
- The Canvas Pushpin module uses the HTML5 canvas to create pushpins. This allows us to create complex pushpins through code without the need for an image based icon.
- If embedding spatial data into an app it is important to remember that updating the data requires creating an update in the Windows Store. This can sometimes takes a couple of weeks to get released.
- Only embed small spatial data files in your application. The larger the embedded file is the more disk space your application will use on the users device. This could be an issue on devices that have a small amount of storage.
- Only load the data you need when you need it. This makes for a much better user experience and better overall performance of the application. If you have a lot of data spread over a large area consider only loading in the data that is in the users map view and loading in additional data as the user navigates the map.

- Minimize the amount of data and compress it if possible. This reduces the bandwidth used by the application and also reduces download times.
- The Bing Spatial Data Services provides an easy way to upload and expose your data as a REST service complete with several common spatial queries such as find nearby, find by bounding box, find along route, find within a geometry, and find by property.
- SkyDrive is a great way to allow your users to store files in the cloud and share them.
- SQLite is a local database that is self-contained, server-less, requires zero-configuration, and is a transactional SQL database that you can use in your Windows Store apps. Using SQLite to manage your data locally is much more efficient than using flat files such as ESRI Shapefiles and XML files.
- When using SQLite store spatial data as Well Known Binary instead of Well Known Text. Well Known Binary takes less time to parse than Well Known Text and the SQLite database will also be smaller.
- A static tile layer is one in which the tiles are created ahead of time and are stored on a server just like any other image file.
- MapCruncher is a Microsoft Research project that makes it easy to cross reference an image with a set of locations on a map and then turn the image into a tile layer. You can download it from here: <http://bit.ly/1aAAkvn> and find a great tutorial on how to use it here: <http://bit.ly/1i1HZu1>.
- Dynamic tile layers are a great way to overlay large or constantly changing data as a tile layer on the map and have good performance.
- Reducing the resolution of the data in a dynamic tile layer at higher zoom levels can really help with performance when drawing the data on a tile. If your data is stored in side of SQL Server 2008, 2012 or Azure you can use the **Reduce** method on the spatial data to reduce its resolution.
- Windows Azure Mobile Services allow you to quickly create connected apps that make it easy to make use of cloud based storage, authenticate users, and send push notifications. Documentation: <http://bit.ly/1aVryy9>.

Chapter 8: Drawing on the Map

Using maps to display information is a pretty standard task but at some point that information needs to be created. In this chapter we are going to develop a simple drawing app that allows us to draw pushpins, polylines, and polygons on the map using the mouse or touch events. To make things a bit more useful we will also include sliders to change the color, opacity and line width of the shapes. We will also include the ability to delete shapes as well.

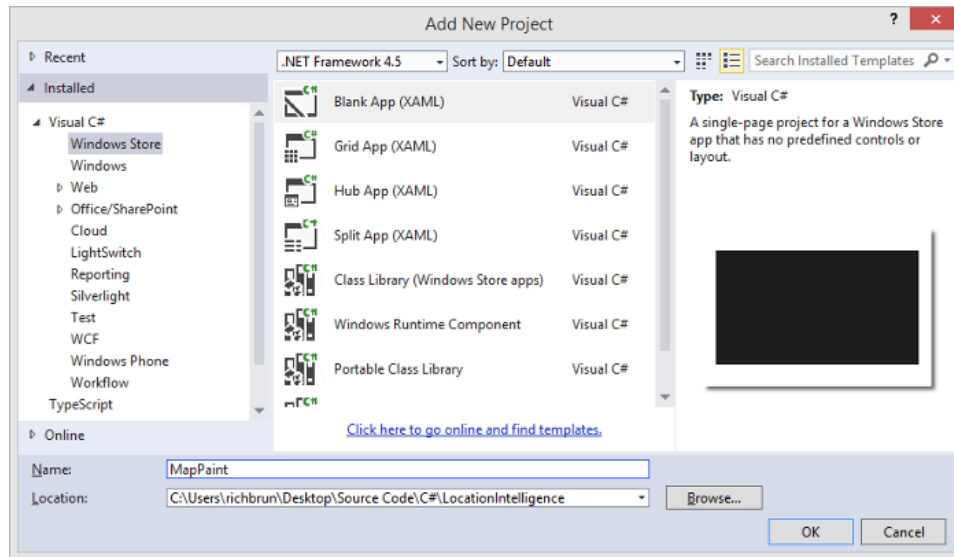
The Windows Store design pattern guide often recommends hiding buttons and controls in the app bars or side panel. However there are exceptions to this rule. If the buttons or controls are going to be used a lot throughout the life cycle of the app it is better to display them at all times to make them easier to access. This is what Microsoft does with the painting tools in their Fresh Paint app (<http://bit.ly/1cvWHX7>) as you can see in the following screenshot.



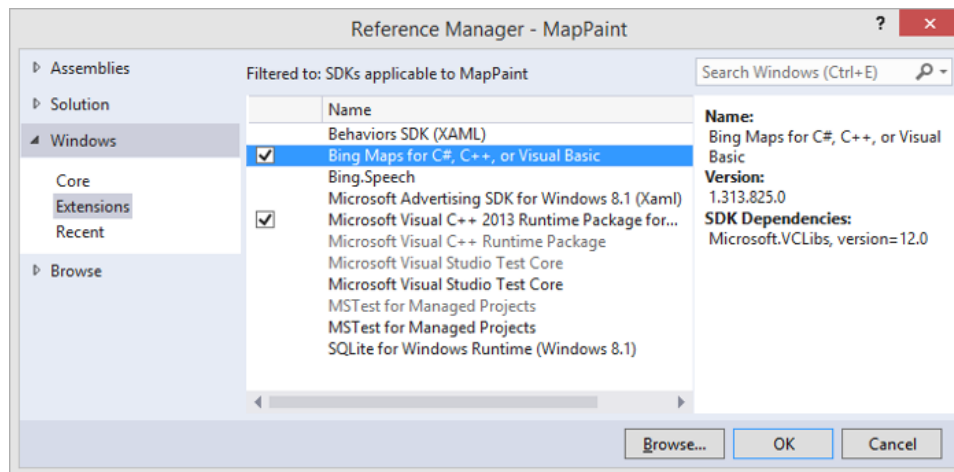
Our application won't be nearly as in depth as the Fresh Paint app but like this app we will place the painting tool buttons in a panel above our canvas (the map). This will make the buttons easy to access and save our user the time of having to open up a panel every time they want to change an option. As mentioned earlier we are going to use sliders to change the color, opacity and the line width settings. The .NET (C#/Visual Basic) Windows Store SDK includes a slider control. Unfortunately the JavaScript Windows Store SDK does not have a slider control, however this can easily be taken care of using jQuery. JQuery (<http://jquery.com/>) is a very popular JavaScript framework that brings a lot of useful JavaScript functions to your application. One branch of jQuery is called jQuery UI (<http://jqueryui.com/>) and includes a number of useful controls such as a slider. We will make use of this control in the JavaScript version of this application. If you are familiar with jQuery and have already tried to get it to work inside of your Windows Store application, chances are you encountered a number of errors. Version 1.x of jQuery makes use of some unsupported JavaScript functions which throw a number of security related errors. This is corrected with version 2.0 of jQuery. One nice thing about jQuery UI is that you can choose between several different themes or even create your own to customize the look of the various controls it provides. In this application I've chosen to use the **smoothness** theme as it looks similar to the .NET slider control.

Creating the base project

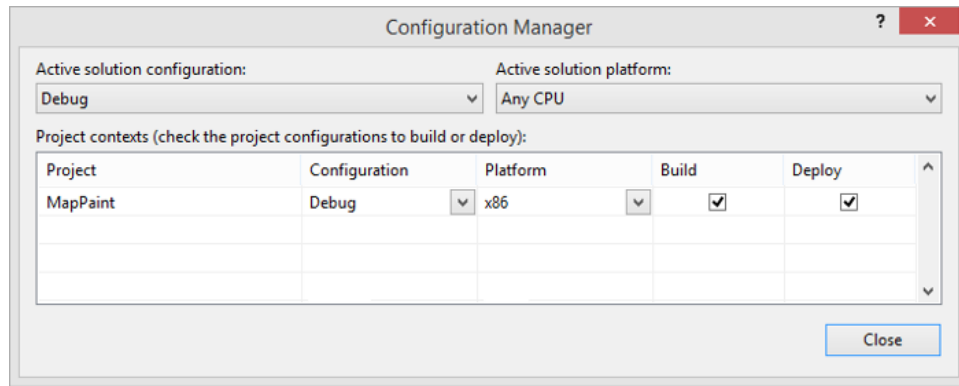
To get started open up Visual Studio and create a new project in your preferred language; JavaScript, C# or Visual Basic. Select the **Blank App** template and call the application **MapPaint** and press **OK**.



Add a reference to the Bing Maps SDK. To do this, right click on the **References** folder and press **Add Reference**. Select **Windows** -> **Extensions** select **Bing Maps for C#, C++ and Visual Basic** or **Bing Maps for JavaScript**. If you do not see this option ensure that you have installed the Bing Maps SDK for Windows Store apps. While you are here, if using C# or Visual Basic, add a reference to the **Microsoft Visual C++ Runtime Package** as this is required by the Bing Maps SDK.



If you are using C# or Visual Basic you may notice that there is a little yellow indicator on the references that you just added. The reason for this is that in the C++ runtime package you have to set the **Active solution platform** in Visual Studio to one of the following options; **ARM**, **x86** or **x64**. To do this, right click on the Solution folder and select **Properties**. Then go to **Configuration Properties -> Configuration**. Find your project and under the **Platform** column set the target platform to x86. Press OK and the yellow indicator should disappear from our references.

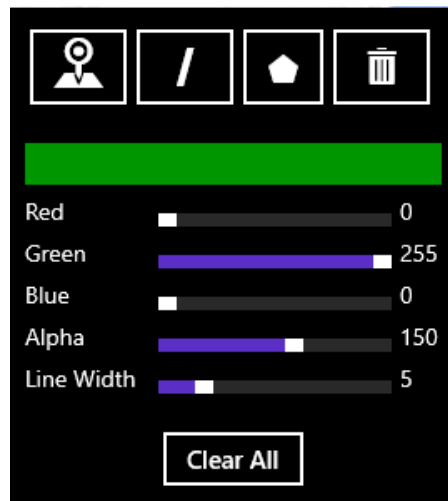


If you are using JavaScript right click on the **js** folder and select **Add -> New Item**. Create two new JavaScript files called **MapPaint.js** and **PaintManager.js**. We will put all our JavaScript for this application in these files to keep things clean. Next download a copy of jQuery 2.0 or above along with jQuery UI or retrieve these files from the sample code project included with this book. Copy the **jquery-2.0.3.js** and **jquery-ui.1.10.3.custom.js** files to the **js** folder. If you have chosen the smoothness theme you should find a folder called **smoothness** in the jQuery UI download. This folder will include a number of CSS style sheets and images. Copy this folder to the **css** folder of the project.

Tip: When selecting a version of jQuery to download you have the option between a compressed and uncompressed version. Since this JavaScript file will be hosted inside of the app it is best to choose the uncompressed version as this will allow you to walk through each line of code when debugging. Since JavaScript files are byte-code cached in Windows Store apps the compressed version of jQuery doesn't offer any advantage other than possibly a slightly smaller project size.

Adding the Map and Paint Control Panel

The painting control panel will consist of a number of buttons and sliders that are displayed on top of the map. We will make the paint control panel will look like this.



If using JavaScript open the **default.html** file and update the HTML with the following. This will add references to all the required CSS and JavaScript files and will also create a div for the map and the HTML for the paint control panel.

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>MapPaint</title>

  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
  <script src="//Microsoft.WinJS.2.0/js/base.js"></script>
  <script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

  <!-- JQuery UI Style Sheet -->
  <link href="css/smoothness/jquery-ui-1.10.3.custom.css" rel="stylesheet">

  <!-- MapPaint references -->
  <link href="/css/default.css" rel="stylesheet" />
  <script src="/js/default.js"></script>

  <!-- Bing Map Control references -->
  <script type="text/javascript" src="ms-
appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>

  <!-- Our Bing Maps JavaScript Code -->
  <script src="/js/MapPaint.js"></script>
  <script src="/js/PaintManager.js"></script>

  <script src="js/jquery-2.0.3.js"></script>
  <script src="js/jquery-ui-1.10.3.custom.js"></script>
</head>
<body>
  <div id="myMap"></div>

  <div id="palette">
    <div class="drawingBtns">
      <button rel="point">&#xE1C4;</button>
      <button rel="line">&#xE199;</button>
      <button rel="polygon">&#x2B1F;</button>
      <button rel="erase">&#xE107;</button>
    </div>

    <table class="sliderTable">
      <tr>
        <td colspan="3"><div id="swatch"></div></td>
      </tr>
      <tr>
        <td>Red</td>
        <td width="130px"><div id="red" rel="rLabel"></div></td>
        <td width="25px"><div id="rLabel"></div></td>
      </tr>
      <tr>
        <td>Green</td>
        <td><div id="green" rel="gLabel"></div></td>
        <td><div id="gLabel"></div></td>
      </tr>
      <tr>
        <td>Blue</td>
        <td><div id="blue" rel="bLabel"></div></td>
        <td><div id="bLabel"></div></td>
      </tr>
      <tr>
        <td>Alpha</td>

```

```

        <td><div id="alpha" rel="aLabel"></div></td>
        <td><div id="aLabel"></div></td>
    </tr>
    <tr>
        <td>Line Width</td>
        <td><div id="lineWidth"></div></td>
        <td><div id="lwLabel"></div></td>
    </tr>
</table>

    <button id="clearBtn">Clear All</button>
</div>
</body>
</html>

```

Next open the **default.css** file and update it with the following CSS styles. These styles will be used to make our paint control panel look like the image from earlier.

```

#palette {
    position:absolute;
    top:75px;
    right:20px;
    background-color:black;
    padding:10px;
    width:250px;
    height:265px;
}

.drawingBtns button {
    font-family:Segoe UI Symbol;
    font-size:24px;
    min-width:57px;
}

.toggleOn {
    background-color:white;
    color:black;
}

.sliderTable {
    margin-top:10px;
    width:100%;
    font-size:14px;
}

#red, #green, #blue, #alpha, #lineWidth {
    float: left;
    clear: left;
    width: 120px;
    margin: 8px 5px;
}

#swatch {
    width: 100%;
    height: 25px;
    border:2px solid white;
}

#clearBtn {
    margin:10px 0 0 75px;
}

```

```

.ui-slider .ui-slider-handle {
    width: 5px;
    height: 12px;
}

.ui-slider-horizontal {
    height: 5px;
}

/* Bug fix for mouse events on Polylines and Polygons in IE11 */
.MicrosoftMapDrawing, .MapPushpinBase {
    pointer-events: auto !important;
}

```

If you are using C# or Visual Basic open the **MainPage.xaml** file and update it with the following XAML.

```

<Page
    x:Class="MapPaint.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MapPaint"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:m="using:Bing.Maps"
    mc:Ignorable="d">

    <Page.Resources>
        <Style TargetType="ToggleButton">
            <Setter Property="FontFamily" Value="Segoe UI Symbol"/>
            <Setter Property="FontSize" Value="24"/>
        </Style>

        <Style TargetType="TextBlock">
            <Setter Property="FontSize" Value="14"/>
            <Setter Property="VerticalAlignment" Value="Bottom"/>
        </Style>
    </Page.Resources>

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <m:Map Name="MyMap" Credentials="YOUR_BING_MAPS_KEY"/>

        <StackPanel Margin="0,75,20,0" Background="Black"
            VerticalAlignment="Top" HorizontalAlignment="Right">

            <StackPanel Orientation="Horizontal" Margin="10">
                <ToggleButton Content="⬮" Name="PointBtn" Tag="point"
                    Tapped="DrawBtn_Tapped"/>
                <ToggleButton Content="⬮" Name="LineBtn" Tag="line"
                    Tapped="DrawBtn_Tapped"/>
                <ToggleButton Content="⬮" Name="PolygonBtn" Tag="polygon"
                    Tapped="DrawBtn_Tapped"/>
                <ToggleButton Content="⬮" Name="EraseBtn" Tag="erase"
                    Tapped="DrawBtn_Tapped"/>
            </StackPanel>

            <Grid Width="250" Height="150" Margin="10 0">
                <Grid.RowDefinitions>
                    <RowDefinition/>
                    <RowDefinition/>
                </Grid.RowDefinitions>
            </Grid>
        </StackPanel>
    </Grid>

```

```

        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="80"/>
        <ColumnDefinition/>
        <ColumnDefinition Width="5"/>
        <ColumnDefinition Width="25"/>
    </Grid.ColumnDefinitions>

    <Border Background="White" BorderThickness="2" BorderBrush="White"
Grid.ColumnSpan="4">
        <Rectangle Name="ShapeColor" Height="40" HorizontalAlignment="Stretch"/>
    </Border>

    <TextBlock Text="Red" Grid.Row="1"/>
    <Slider Name="RSlider" LargeChange="5" Maximum="255"
ValueChanged="Slider_ValueChanged" Grid.Row="1" Grid.Column="1"/>
    <TextBlock Text="{Binding ElementName=RSlider,Path=Value}" Grid.Row="1"
Grid.Column="3" />

    <TextBlock Text="Green" Grid.Row="2"/>
    <Slider Name="GSlider" LargeChange="5" Maximum="255"
ValueChanged="Slider_ValueChanged" Grid.Row="2" Grid.Column="1"/>
    <TextBlock Text="{Binding ElementName=GSlider,Path=Value}" Grid.Row="2"
Grid.Column="3"/>

    <TextBlock Text="Blue" Grid.Row="3"/>
    <Slider Name="BSlider" LargeChange="5" Maximum="255"
ValueChanged="Slider_ValueChanged" Grid.Row="3" Grid.Column="1"/>
    <TextBlock Text="{Binding ElementName=BSlider,Path=Value}" Grid.Row="3"
Grid.Column="3"/>

    <TextBlock Text="Alpha" Grid.Row="4"/>
    <Slider Name="ASlider" LargeChange="5" Maximum="255"
ValueChanged="Slider_ValueChanged" Grid.Row="4" Grid.Column="1"/>
    <TextBlock Text="{Binding ElementName=ASlider,Path=Value}" Grid.Row="4"
Grid.Column="3"/>

    <TextBlock Text="Line Width" Grid.Row="5"/>
    <Slider Name="WidthSlider" LargeChange="1" Minimum="1" Maximum="25"
ValueChanged="Slider_ValueChanged" Grid.Row="5" Grid.Column="1"/>
    <TextBlock Text="{Binding ElementName=WidthSlider,Path=Value}" Grid.Row="5"
Grid.Column="3"/>
    </Grid>

    <Button Content="Clear All" Tapped="ClearBtn_Tapped"
HorizontalAlignment="Center" Margin="10"/>
    </StackPanel>
</Grid>
</Page>

```

In the code behind we will need to add code to load the map, initialize the sliders and add event handlers for buttons. Open the **MapPaint.js** or the **MainPage.xaml.cs** file and update it with the following code.

JavaScript

```
(function () {
```

```

var map;

function initialize() {
    Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: GetMap });

    //Create color sliders
    $("#red, #green, #blue, #alpha").slider({
        orientation: "horizontal",
        range: "min",
        max: 255,
        value: 0,
        slide: colorChanged,
        change: colorChanged
    });

    $("#red").slider("value", 0);
    $("#green").slider("value", 255);
    $("#blue").slider("value", 0);
    $("#alpha").slider("value", 150);

    //Create line width slider
    $("#lineWidth").slider({
        orientation: "horizontal",
        min: 1,
        max: 25,
        value: 0,
        slide: lineWidthChanged,
        change: lineWidthChanged
    });

    $("#lineWidth").slider("value", 5);
}

function colorChanged(e, ui) {
}

function lineWidthChanged(e, ui) {
}

function GetMap() {
    map = new Microsoft.Maps.Map(document.getElementById("myMap"), {
        credentials: "YOUR_BING_MAPS_KEY"
    });
}

document.addEventListener("DOMContentLoaded", initialize, false);
})();

```

C#

```

using System;
using Windows.UI;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;

namespace MapPaint
{
    public sealed partial class MainPage : Page

```

```

{
    public MainPage()
    {
        this.InitializeComponent();

        GSlider.Value = 255;
        ASlider.Value = 150;
        WidthSlider.Value = 5;
    }

    private void DrawBtn_Tapped(object sender, TappedRoutedEventArgs e)
    {
    }

    private void ClearBtn_Tapped(object sender, TappedRoutedEventArgs e)
    {
    }

    private void Slider_ValueChanged(object sender, RangeBaseValueChangedEventArgs
e)
    {
    }
}

```

Visual Basic

```

Imports Bing.Maps
Imports Windows.UI

Public NotInheritable Class MainPage
    Inherits Page

    Public Sub New()
        Me.InitializeComponent()

        GSlider.Value = 255
        ASlider.Value = 150
        WidthSlider.Value = 5
    End Sub

    Private Sub DrawBtn_Tapped(sender As Object, e As TappedRoutedEventArgs)
    End Sub

    Private Sub Slider_ValueChanged(sender As Object, e As
RangeBaseValueChangedEventArgs)
    End Sub

    Private Sub ClearBtn_Tapped(sender As Object, e As TappedRoutedEventArgs)
    End Sub
End Class

```

If you run the application you will a map displayed with the paint control panel in the tip right corner of the map. Note that none of the buttons and sliders will do anything yet as we haven't added the logic for these controls yet.



Creating a Paint Manager

To draw shapes on the map we will need to make use of a number of touch, mouse and map events. Adding all this logic in line with the rest of the code for our application would make things pretty messy and hard to follow. It would also make it very hard to reuse the code in another project. To make things easier we will create a class called **PaintManager** which will take in a reference to the map and manage all the events needed to draw on the map. In the .NET version of this class a **MapShapeLayer** and a **MapLayer** will be created for rendering the shapes and pushpins on the map. In the JavaScript version a single **EntityCollection** will be used to render the shapes and pushpins on the map. The **PaintManager** will provide access to these layers. This will allow us to import or retrieve these shapes from the layer. This may be useful in future applications if you want to include an import/export functionality to the app. The **PaintManager** will also expose properties for the current color and line width to be used. These methods will be added to **PaintManager** class; **StartDrawing**, **EndDrawing** and **Clear**. The **StartDrawing** method will take in a string that indicates the drawing mode that has been selected. The supported drawing modes include; **point**, **line**, **polygon**, and **erase**. When the **StartDrawing** method is called the required touch, mouse and map events will be added to the map. The JavaScript version will disable panning and zooming of the map using map options. The .NET version of the app will disable panning by storing the center of the map and using the **ViewChanged** event on the map to update the map center to the stored value. By doing this the user will be able to click and drag new points they want to add to the shape.

When the **erase** drawing mode is enabled a click event will be added to all shapes on the map. If the user clicks on any of the shapes they will be deleted. At the time of writing this book the JavaScript version of Bing Maps doesn't appear to support clicking of polygons and polylines. However we will still wire these events up so that if this is corrected in the future we won't need to update our app to make use of this feature.

If using C# or Visual Basic create a new class called **PaintManager** by right clicking on the project and selecting **Add -> New Item**. Select the Class template and name it **PaintManager.cs**. If using JavaScript the **PaintManager** class logic will be added to the **PaintManager.js** file. Open the appropriate file for the **PaintManager** class and add the following code.

JavaScript

```

var PaintManager = function (map) {

    var shapeLayer, currentShape, center, currentLocation, coordIdx, isDrawing = false,
    drawingMode = null;

    var color = new Microsoft.Maps.Color(150,0,255,0);
    var lineWidth = 5;

    var eventIds = [];

    function init(){
        shapeLayer = new Microsoft.Maps.EntityCollection();
        map.entities.push(shapeLayer);
    }

    /**
     * Public Property set/get
     */

    this.setColor = function(c){
        color = c;
    };

    this.getColor = function(){
        return color;
    };

    this.setLineWidth = function(w){
        lineWidth = w;
    };

    this.getLineWidth = function(){
        return lineWidth;
    };

    this.getShapeLayer = function(){
        return shapeLayer;
    };

    this.getDrawingMode = function () {
        return drawingMode;
    };

    /**
     * Public Methods
     */

    this.startDrawing = function (dm) {
        //end any previous drawing state
        this.endDrawing();

        drawingMode = dm;

        if (drawingMode == "erase")
        {
            for (var i = 0; i < shapeLayer.getLength() ; i++) {
                var s = shapeLayer.get(i);
                eventIds.push(Microsoft.Maps.Events.addHandler(s, 'click', eraseShape));
            }
        }
    }
}

```

```

    }
    else
    {
        eventIds.push(Microsoft.Maps.Events.addHandler(map, 'mouseup',
mapPointReleased));
        eventIds.push(Microsoft.Maps.Events.addHandler(map, 'mousemove',
mapPointerMoved));

        //disable panning and zooming
        map.setOptions({ disablePanning: true, disableZooming: true });
    }
};

this.endDrawing = function () {
    for (var i = 0; i < eventIds.length; i++) {
        Microsoft.Maps.Events.removeHandler(eventIds[i]);
    }

    eventIds = [];

    if (currentShape != null) {
        var locs = currentShape.getLocations();

        if (locs.length > coordIdx) {
            locs.pop();
        }

        switch (drawingMode) {
            case "polygon":
                //Close the polygon
                locs.push(locs[0]);
                break;
        }

        currentShape.setLocations(locs);

        currentShape = null;
    }

    //re-enable panning and zooming
    map.setOptions({ disablePanning: false, disableZooming: false });

    drawingMode = null;
    isDrawing = false;
};

this.clear = function () {
    shapelayer.clear();
};

/*****
* Private Methods
*****/

function mapPointReleased(e) {
    var point = new Microsoft.Maps.Point(e.getX(), e.getY());

    //Convert the point pixel to a Location coordinate
    currentLocation = map.tryPixelToLocation(point);

    //Initialize drawing state

```

```

    if (!isDrawing) {
        center = map.getCenter();

        switch (drawingMode) {
            case "point":
                isDrawing = true;
                break;
            case "line":
                currentShape = new Microsoft.Maps.Polyline([currentLocation,
currentLocation],
                    {
                        strokeColor: color,
                        strokeThickness: lineWidth
                    });

                coordIdx = 0;
                break;
            case "polygon":
                currentShape = new Microsoft.Maps.Polygon([currentLocation,
currentLocation, currentLocation],
                    {
                        fillColor: color
                    });

                coordIdx = 0;
                break;
            default:
                break;
        }

        if (currentShape != null) {
            shapeLayer.push(currentShape);
            isDrawing = true;
        }
    }

    //Continue drawing shape with each press
    switch (drawingMode) {
        case "point":
            var pin = new Microsoft.Maps.Pushpin(currentLocation);
            shapeLayer.push(pin);
            break;
        case "line":
        case "polygon":
            var locs = currentShape.getLocations();

            if (coordIdx < locs.length) {
                locs[coordIdx] = currentLocation;
            } else {
                locs.push(currentLocation);
            }

            currentShape.setLocations(locs);
            coordIdx++;
            break;
        default:
            break;
    }
}

function mapPointerMoved(e) {

```

```

    if (isDrawing) {
        var point = new Microsoft.Maps.Point(e.getX(), e.getY());
        var tempLocation = map.tryPixelToLocation(point);

        switch (drawingMode) {
            case "line":
            case "polygon":
                var locs = currentShape.getLocations();

                //Update the last coordinate in shape to preview new point
                locs[coordIdx] = tempLocation;

                currentShape.setLocations(locs);
                break;
        }
    }
}

function eraseShape(e) {
    if (e.target) {
        shapeLayer.remove(e.target);
    }
}

init();
};

```

C#

```

using Bing.Maps;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Windows.UI;
using Windows.UI.Xaml.Media;

namespace MapPaint
{
    public class PaintManager
    {
        #region Private Properties

        private Map _map;
        private MapShapeLayer shapeLayer;
        private MapLayer pinLayer;
        private MapMultiPoint currentShape;

        private Location center;
        private Location currentLocation;

        private int coordIdx;

        private bool isDrawing;
        private string _drawingMode;

        #endregion

        #region Constructor

```

```

public PaintManager(Map map)
{
    _map = map;

    shapeLayer = new MapShapeLayer();
    _map.ShapeLayers.Add(shapeLayer);

    pinLayer = new MapLayer();
    _map.Children.Add(pinLayer);
}

#endregion

#region Public Properties

public Color Color { get; set; }

public double Linewidth { get; set; }

public MapShapeLayer ShapeLayer
{
    get
    {
        return shapeLayer;
    }
}

public MapLayer PinLayer
{
    get
    {
        return pinLayer;
    }
}

#endregion

#region Public Methods

public void StartDrawing(string drawingMode)
{
    //end any previous drawing state
    EndDrawing();
    _drawingMode = drawingMode;
    center = _map.Center;

    if (string.Compare(_drawingMode, "erase",
StringComparison.OrdinalIgnoreCase) == 0)
    {
        foreach (var s in shapeLayer.Shapes)
        {
            s.Tapped += EraseShape_Tapped;
        }

        foreach (var p in pinLayer.Children)
        {
            p.Tapped += ErasePin_Tapped;
        }
    }
    else

```

```

        {
            _map.PointerReleasedOverride += MapPointerReleased;
            _map.PointerMovedOverride += MapPointerMoved;
            _map.ViewChanged += MapViewChanged;
        }
    }

    public void EndDrawing()
    {
        if (string.Compare(_drawingMode, "erase",
StringComparison.OrdinalIgnoreCase) == 0)
        {
            foreach (var s in shapeLayer.Shapes)
            {
                s.Tapped -= EraseShape_Tapped;
            }

            foreach (var p in pinLayer.Children)
            {
                p.Tapped -= ErasePin_Tapped;
            }
        }
        else
        {
            if (currentShape != null)
            {
                if (currentShape.Locations.Count > coordIdx)
                {
                    currentShape.Locations.RemoveAt(coordIdx);
                }

                switch (_drawingMode)
                {
                    case "polygon":
                        //Close the polygon
                        currentShape.Locations.Add(currentShape.Locations[0]);
                        break;
                }

                currentShape = null;
            }

            _map.PointerReleasedOverride -= MapPointerReleased;
            _map.PointerMovedOverride -= MapPointerMoved;
            _map.ViewChanged -= MapViewChanged;
        }

        _drawingMode = string.Empty;
        isDrawing = false;
    }

    public void Clear()
    {
        EndDrawing();
        shapeLayer.Shapes.Clear();
        pinLayer.Children.Clear();
    }

    #endregion

    #region Private Methods

```



```

private void MapPointerMoved(object sender,
Windows.UI.Xaml.Input.PointerRoutedEventArgs e)
{
    if (isDrawing)
    {
        var pointerPosition = e.GetCurrentPoint(_map);
        Location tempLocation;

        if (_map.TryPixelToLocation(pointerPosition.Position, out tempLocation))
        {
            switch (_drawingMode)
            {
                case "line":
                case "polygon":
                    //Update the last coordinate in shape to preview new point
                    currentShape.Locations[coordIdx] = tempLocation;
                    break;
            }
        }
    }
}

private void MapViewChanged(object sender, ViewChangedEventArgs e)
{
    if (isDrawing)
    {
        //Reset the map center to the stored center value.
        //This prevents the map from panning when we drag across it.
        _map.Center = center;
    }
}

private void MapPointerReleased(object sender,
Windows.UI.Xaml.Input.PointerRoutedEventArgs e)
{
    var pointerPosition = e.GetCurrentPoint(_map);

    //Convert the point pixel to a Location coordinate
    if (_map.TryPixelToLocation(pointerPosition.Position, out currentLocation))
    {
        //Initialize drawing state
        if (!isDrawing)
        {
            switch (_drawingMode)
            {
                case "point":
                    isDrawing = true;
                    break;
                case "line":
                    currentShape = new MapPolyline()
                    {
                        Color = Color,
                        Width = LineWidth,
                        Locations = new LocationCollection()
                    };
                    break;
                case "polygon":
                    currentShape = new MapPolygon()
                    {
                        FillColor = Color,

```

```

        Locations = new LocationCollection()
    };
    break;
default:
    break;
}

if (currentShape != null)
{
    currentShape.Locations.Add(currentLocation);
    shapeLayer.Shapes.Add(currentShape);

    coordIdx = 0;
    isDrawing = true;
}
}

//Continue drawing shape with each press
switch (_drawingMode)
{
    case "point":
        var pin = new Pushpin()
        {
            Background = new SolidColorBrush(Color)
        };

        MapLayer.SetPosition(pin, currentLocation);
        pinLayer.Children.Add(pin);
        break;
    case "line":
    case "polygon":
        currentShape.Locations.Add(currentLocation);
        coordIdx++;
        break;
    default:
        break;
}
}
}

private void ErasePin_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
    var pin = sender as Windows.UI.Xaml.UIElement;
    pinLayer.Children.Remove(pin);
}

private void EraseShape_Tapped(object sender,
Windows.UI.Xaml.Input.TappedRoutedEventArgs e)
{
    var shape = sender as MapShape;
    shapeLayer.Shapes.Remove(shape);
}

#endregion
}
}

```

```

Imports Bing.Maps
Imports Windows.UI

Public Class PaintManager

#Region "Private Properties"

    Private _map As Map
    Private _shapeLayer As MapShapeLayer
    Private _pinLayer As MapLayer
    Private currentShape As MapMultiPoint

    Private center As Location
    Private currentLocation As Location

    Private coordIdx As Integer

    Private isDrawing As Boolean
    Private _drawingMode As String

#End Region

#Region "Constructor"

    Public Sub New(map As Map)
        _map = map

        _shapeLayer = New MapShapeLayer()
        _map.ShapeLayers.Add(_shapeLayer)

        _pinLayer = New MapLayer()
        _map.Children.Add(_pinLayer)
    End Sub

#End Region

#Region "Public Properties"

    Public Property Color As Color

    Public Property LineWidth As Double

    Public ReadOnly Property ShapeLayer() As MapShapeLayer
        Get
            Return _shapeLayer
        End Get
    End Property

    Public ReadOnly Property PinLayer() As MapLayer
        Get
            Return _pinLayer
        End Get
    End Property

#End Region

#Region "Public Methods"

    Public Sub StartDrawing(drawingMode As String)
        'end any previous drawing state

```

```

EndDrawing()
_drawingMode = drawingMode
center = _map.Center

If String.Compare(_drawingMode, "erase", StringComparison.OrdinalIgnoreCase) = 0
Then
    For Each s As MapShape In _shapeLayer.Shapes
        AddHandler s.Tapped, AddressOf EraseShape_Tapped
    Next

    For Each p As UIElement In _pinLayer.Children
        AddHandler p.Tapped, AddressOf ErasePin_Tapped
    Next
Else
    AddHandler _map.PointerReleasedOverride, AddressOf MapPointerReleased
    AddHandler _map.PointerMovedOverride, AddressOf MapPointerMoved
    AddHandler _map.ViewChanged, AddressOf MapViewChanged
End If
End Sub

Public Sub EndDrawing()
    If String.Compare(_drawingMode, "erase", StringComparison.OrdinalIgnoreCase) = 0
Then
        For Each s As MapShape In _shapeLayer.Shapes
            RemoveHandler s.Tapped, AddressOf EraseShape_Tapped
        Next

        For Each p As UIElement In _pinLayer.Children
            RemoveHandler p.Tapped, AddressOf ErasePin_Tapped
        Next
    Else
        If currentShape IsNot Nothing Then
            If currentShape.Locations.Count > coordIdx Then
                currentShape.Locations.RemoveAt(coordIdx)
            End If

            Select Case _drawingMode
            Case "polygon"
                'Close the polygon
                currentShape.Locations.Add(currentShape.Locations(0))
            Exit Select
            End Select

            currentShape = Nothing
        End If

        RemoveHandler _map.PointerReleasedOverride, AddressOf MapPointerReleased
        RemoveHandler _map.PointerMovedOverride, AddressOf MapPointerMoved
        RemoveHandler _map.ViewChanged, AddressOf MapViewChanged
    End If

    _drawingMode = String.Empty
    isDrawing = False
End Sub

Public Sub Clear()
    EndDrawing()
    _shapeLayer.Shapes.Clear()
    _pinLayer.Children.Clear()
End Sub

```

```
#End Region
```

```
#Region "Private Methods"
```

```
Private Sub MapPointerMoved(sender As Object, e As
Windows.UI.Xaml.Input.PointerRoutedEventArgs)
    If isDrawing Then
        Dim pointerPosition = e.GetCurrentPoint(_map)
        Dim tempLocation As Location

        If _map.TryPixelToLocation(pointerPosition.Position, tempLocation) Then
            Select Case _drawingMode
                Case "line", "polygon"
                    'Update the last coordinate in shape to preview new point
                    currentShape.Locations(coordIdx) = tempLocation
                Exit Select
            End Select
        End If
    End If
End Sub
```

```
Private Sub MapViewChanged(sender As Object, e As ViewChangedEventArgs)
    If isDrawing Then
        'Reset the map center to the stored center value.
        'This prevents the map from panning when we drag across it.
        _map.Center = center
    End If
End Sub
```

```
Private Sub MapPointerReleased(sender As Object, e As
Windows.UI.Xaml.Input.PointerRoutedEventArgs)
    Dim pointerPosition = e.GetCurrentPoint(_map)

    'Convert the point pixel to a Location coordinate
    If _map.TryPixelToLocation(pointerPosition.Position, currentLocation) Then
        'Initialize drawing state
        If Not isDrawing Then

            Select Case _drawingMode
                Case "point"
                    isDrawing = True
                    Exit Select
                Case "line"
                    Dim line = New MapPolyline()
                    line.Color = Color
                    line.Width = LineWidth
                    line.Locations = New LocationCollection()
                    currentShape = line
                    Exit Select
                Case "polygon"
                    Dim poly = New MapPolygon()
                    poly.FillColor = Color
                    poly.Locations = New LocationCollection()
                    currentShape = poly
                    Exit Select
                Case Else
                    Exit Select
            End Select

            If currentShape IsNot Nothing Then
                currentShape.Locations.Add(currentLocation)
            End If
        End If
    End If
End Sub
```

```

        _shapeLayer.Shapes.Add(currentShape)

        coordIdx = 0
        isDrawing = True
    End If
End If

'Continue drawing shape with each press
Select Case _drawingMode
    Case "point"
        Dim pin = New Pushpin()
        pin.Background = New SolidColorBrush(Color)

        MapLayer.SetPosition(pin, currentLocation)
        _pinLayer.Children.Add(pin)
    Exit Select
    Case "line", "polygon"
        currentShape.Locations.Add(currentLocation)
        coordIdx += 1
    Exit Select
    Case Else
        Exit Select
End Select
End If
End Sub

Private Sub ErasePin_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
    Dim pin = TryCast(sender, Windows.UI.Xaml.UIElement)
    _pinLayer.Children.Remove(pin)
End Sub

Private Sub EraseShape_Tapped(sender As Object, e As
Windows.UI.Xaml.Input.TappedRoutedEventArgs)
    Dim shape = TryCast(sender, MapShape)
    _shapeLayer.Shapes.Remove(shape)
End Sub
#End Region

End Class

```

Bringing Everything Together

At this point we have all the key components we need for our application. The last step is to bring everything together and connect the paint control panel to the paint manager. To do this we must first create an instance of the **PaintManager** class after the map is loaded and store a reference to this class as a variable in our application. Once this is done the event handler for the drawing buttons can be updated to end any current drawing that may be in progress and to start a new drawing mode. The user will be able to toggle the drawing buttons to start and stop the drawing mode. When the color sliders change the preview color palette will be updated with the new color and the paint managers **Color** property updated as well. Similarly when the line width slider is changed the paint managers **LineWidth** property will also be updated. The final functionality to add is the button handler that clears the map. Update the code in the **MapPaint.js** or **MainPage.xaml.cs** file with the following to connect the paint control panel to the **PaintManager** class.

JavaScript

```

(function () {

```

```

var map, paintManager;

function initialize() {
    Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: GetMap });

    //Add click events to drawing buttons.
    $(".drawingBtns > button").click(function () {
        //Check to see if this toggle button is on
        if ($(this).hasClass("toggleOn")) {
            $(this).removeClass("toggleOn");
            paintManager.endDrawing();
        } else {
            //Toggle off all buttons
            $(".drawingBtns > button").removeClass("toggleOn");

            //Toggle on this button
            $(this).addClass("toggleOn");

            //Get drawing mode
            var drawMode = $(this).attr("rel");
            paintManager.startDrawing(drawMode);
        }
    });

    //Create color sliders
    $("#red, #green, #blue, #alpha").slider({
        orientation: "horizontal",
        range: "min",
        max: 255,
        value: 0,
        slide: colorChanged,
        change: colorChanged
    });

    $("#red").slider("value", 0);
    $("#green").slider("value", 255);
    $("#blue").slider("value", 0);
    $("#alpha").slider("value", 150);

    //Create line width slider
    $("#lineWidth").slider({
        orientation: "horizontal",
        min: 1,
        max: 25,
        value: 0,
        slide: lineWidthChanged,
        change: lineWidthChanged
    });

    $("#lineWidth").slider("value", 5);

    $("#clearBtn").click(function () {
        paintManager.clear();

        //Toggle off all buttons
        $(".drawingBtns > button").removeClass("toggleOn");
    });
}

function colorChanged(e, ui) {
    var r = $("#red").slider("value"),

```

```

        g = $("#green").slider("value"),
        b = $("#blue").slider("value"),
        a = $("#alpha").slider("value");

    var color = new Microsoft.Maps.Color(a, r, g, b);

    if (paintManager) {
        paintManager.setColor(color);
    }

    $("#swatch").css("background-color", "#" + color.toHex());
    $("##" + $(this).attr("rel")).html(ui.value);
}

function lineWidthChanged(e, ui) {
    if (paintManager) {
        paintManager.setLineWidth(ui.value);
    }

    $("#lwLabel").html(ui.value);
}

function GetMap() {
    map = new Microsoft.Maps.Map(document.getElementById("myMap"), {
        credentials: "YOUR_BING_MAPS_KEY"
    });

    paintManager = new PaintManager(map);
}

document.addEventListener("DOMContentLoaded", initialize, false);
})();

```

C#

```

using System;
using Windows.UI;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;

namespace MapPaint
{
    public sealed partial class MainPage : Page
    {
        #region Private Properties

        private PaintManager paintManager;

        #endregion

        #region Constructor

        public MainPage()
        {
            this.InitializeComponent();

            paintManager = new PaintManager(MyMap);
        }
    }
}

```



```

        GSlider.Value = 255;
        ASlider.Value = 150;
        WidthSlider.Value = 5;
    }

    #endregion

    #region Event Handlers

    private void DrawBtn_Tapped(object sender, TappedRoutedEventArgs e)
    {
        var btn = sender as ToggleButton;
        var drawMode = btn.Tag as string;

        paintManager.EndDrawing();

        if (btn.IsChecked.HasValue && btn.IsChecked.Value)
        {
            //Start the new drawing mode
            paintManager.StartDrawing(drawMode);
        }

        //Sync toggle buttons
        switch (drawMode)
        {
            case "point":
                LineBtn.IsChecked = false;
                PolygonBtn.IsChecked = false;
                EraseBtn.IsChecked = false;
                break;
            case "line":
                PointBtn.IsChecked = false;
                PolygonBtn.IsChecked = false;
                EraseBtn.IsChecked = false;
                break;
            case "polygon":
                PointBtn.IsChecked = false;
                LineBtn.IsChecked = false;
                EraseBtn.IsChecked = false;
                break;
            case "erase":
                PointBtn.IsChecked = false;
                LineBtn.IsChecked = false;
                PolygonBtn.IsChecked = false;
                break;
        }
    }

    private void ClearBtn_Tapped(object sender, TappedRoutedEventArgs e)
    {
        paintManager.Clear();

        PointBtn.IsChecked = false;
        LineBtn.IsChecked = false;
        PolygonBtn.IsChecked = false;
        EraseBtn.IsChecked = false;
    }

    private void Slider_ValueChanged(object sender, RangeBaseValueChangedEventArgs
e)
    {

```

```

        if (paintManager != null)
        {
            byte R, G, B, A;

            A = Convert.ToByte(ASlider.Value);
            R = Convert.ToByte(RSlider.Value);
            G = Convert.ToByte(GSlider.Value);
            B = Convert.ToByte(BSlider.Value);

            paintManager.Color = Color.FromArgb(A, R, G, B);
            paintManager.LineWidth = WidthSlider.Value;

            ShapeColor.Fill = new SolidColorBrush(paintManager.Color);
        }
    }
}
#endregion
}
}

```

Visual Basic

```

Imports Bing.Maps
Imports Windows.UI

Public NotInheritable Class MainPage
    Inherits Page

    #Region "Private Properties"

        Private paintManager As PaintManager

    #End Region

    #Region "Constructor"

        Public Sub New()
            Me.InitializeComponent()

            paintManager = New PaintManager(MyMap)

            GSlider.Value = 255
            ASlider.Value = 150
            WidthSlider.Value = 5
        End Sub

    #End Region

    #Region "Event Handlers"

        Private Sub DrawBtn_Tapped(sender As Object, e As TappedRoutedEventArgs)
            Dim btn = TryCast(sender, ToggleButton)
            Dim drawMode = TryCast(btn.Tag, String)

            paintManager.EndDrawing()

            If btn.IsChecked.HasValue AndAlso btn.IsChecked.Value Then
                'Strat the new drawing mode
                paintManager.StartDrawing(drawMode)
            End If
        End Sub
    End Sub

```

```

'Sync toggle buttons
Select Case drawMode
    Case "point"
        LineBtn.IsChecked = False
        PolygonBtn.IsChecked = False
        EraseBtn.IsChecked = False
    Exit Select
    Case "line"
        PointBtn.IsChecked = False
        PolygonBtn.IsChecked = False
        EraseBtn.IsChecked = False
    Exit Select
    Case "polygon"
        PointBtn.IsChecked = False
        LineBtn.IsChecked = False
        EraseBtn.IsChecked = False
    Exit Select
    Case "erase"
        PointBtn.IsChecked = False
        LineBtn.IsChecked = False
        PolygonBtn.IsChecked = False
    Exit Select
End Select
End Sub

Private Sub Slider_ValueChanged(sender As Object, e As
RangeBaseValueChangedEventArgs)
    If paintManager IsNot Nothing Then
        Dim R As Byte, G As Byte, B As Byte, A As Byte

        A = Convert.ToByte(ASlider.Value)
        R = Convert.ToByte(RSlider.Value)
        G = Convert.ToByte(GSlider.Value)
        B = Convert.ToByte(BSlider.Value)

        paintManager.Color = Color.FromArgb(A, R, G, B)
        paintManager.LineWidth = WidthSlider.Value

        ShapeColor.Fill = New SolidColorBrush(paintManager.Color)
    End If
End Sub

Private Sub ClearBtn_Tapped(sender As Object, e As TappedRoutedEventArgs)
    If paintManager IsNot Nothing Then
        paintManager.Clear()

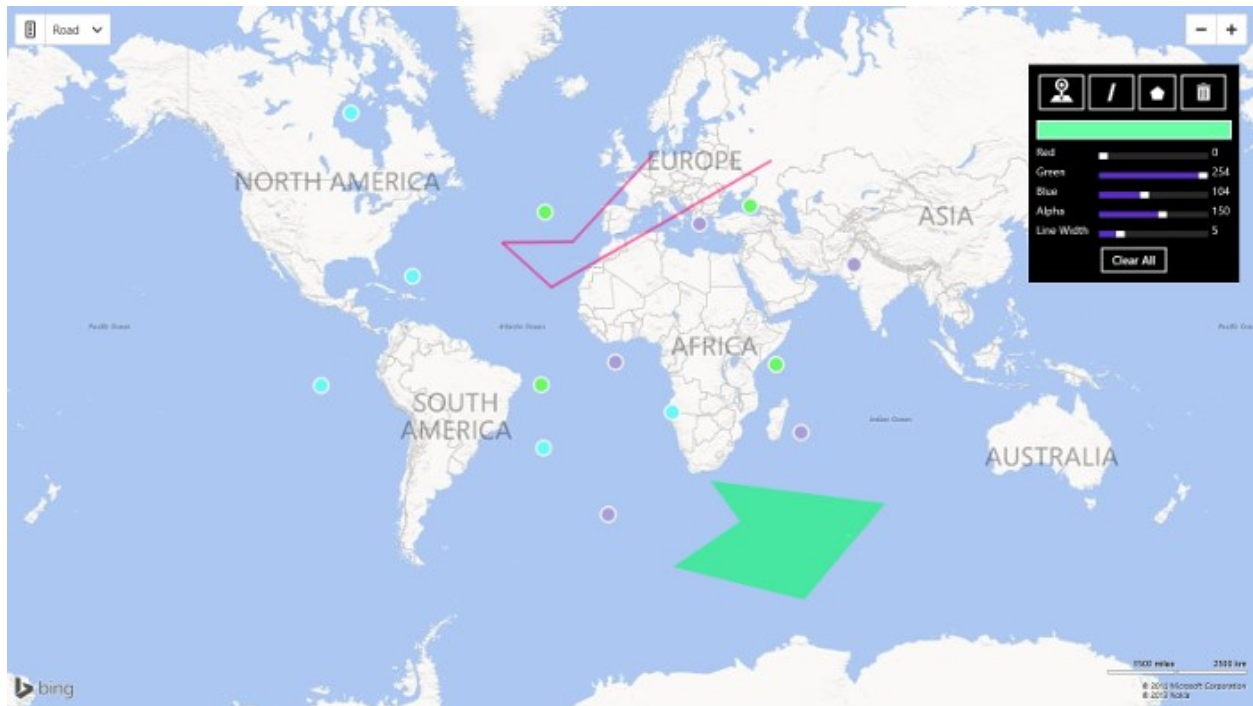
        PointBtn.IsChecked = False
        LineBtn.IsChecked = False
        PolygonBtn.IsChecked = False
        EraseBtn.IsChecked = False
    End If
End Sub

#End Region
End Class

```

At this point the application is complete. The final step is to run and test the application. Press the Debug button or F5 to launch the application. Select a drawing mode and use the mouse or touch events to start drawing on the map.

When you are done simply press the drawing button again to toggle it off or select a different drawing mode. Here is a screenshot of the finished application with some shapes drawn on it.



Taking the application further

This simple drawing application is a good starting point for a lot of other types of applications. Here are some ideas of how you may want to expand upon this application.

- Using the functionality to draw a line on the map can easily be modified to measure distances. Simply calculate the distances between each coordinate using the Haversine formula (included in the SpatialToolbox library).
- Importing or Exporting of common spatial file types can be easily done using the tools in the SpatialToolbox. Simply add or retrieve the shapes from the layer properties on the **PaintManager** class. This would make for an easy way to edit spatial files.
- Most searches are performed using a center point and a radius. Why not use the polygon drawing functionality to provide your users with the ability to draw out custom areas on the map in which they want to search within.
- Add additional editing capabilities such as input textboxes for a title and description for each shape.

Chapter 9: Creating an Augmented Reality App

When many people hear the words “augmented reality” the first thought that comes to mind is the large helmet type video games that never really took off in the 90’s. This was largely due to the high costs and bulky hardware that was required. This is also a bit inaccurate, as these video games were actually “virtual reality” games. Augmented reality (AR) is when an object that is not present in the real world, but appears to be because of a view showing a modified version of reality. Virtual reality is similar, but instead of being in the real world, the user is viewing a simulated version of the world.

The idea behind augmented reality was first envisioned in 1901 by author Lyman Frank Baum in the book titled “The Master Key”. In this book the main character has a pair of glasses that, when worn, would show the type of person someone was (i.e. good or bad). Now if this author’s name sounds familiar to you, but you can’t put your finger out it, he is best known for writing “The Wizard of Oz”.

In recent years augmented reality has become increasingly popular. It has the smartphone to thank for this. With smartphones essentially being small-scaled computers filled with motion sensors, they have become the ideal platform for creating relatively inexpensive augmented reality applications.

Augmented reality apps are often used to show the location of a number of different types of features such as:

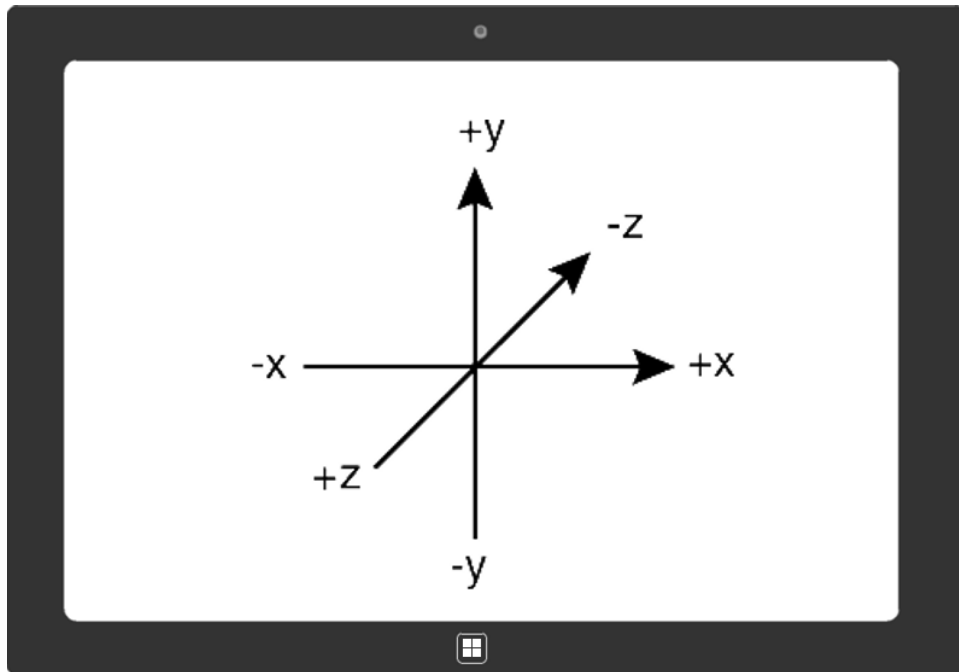
- Points of interest
- Pictures that have been taken around you
- Nearby information from Twitter, Wikipedia, Foursquare, Urban spoon
- Where you parked the car
- Golf course information, such as the location of the pin or hazards

A rear facing camera is often used to display the world behind the device as a streaming video on the screen. This makes the device feel like more of a window into the augmented reality world.

Sensors in Augmented Reality Apps

As we saw in Chapter 2 Microsoft added in a new location and sensors platform in Windows 7. This sensor platform has grown significantly in Windows 8, to the point where many new computers have the same sensors that are found in most smartphones. This means that many of the augmented reality applications that exist for smartphones can also be created as a Windows Store app.

Augmented reality apps rely heavily on sensors. There are many different sensors that can be used. These sensors return a number of different measurements, many of them are relative to the x, y and z axis of the device. When facing the screen of the device, the x-axis runs in a left to right direction, the y-axis runs from the bottom up, and the z-axis runs from the back of the screen through to the front. Here is a diagram showing these axes relative to a device.



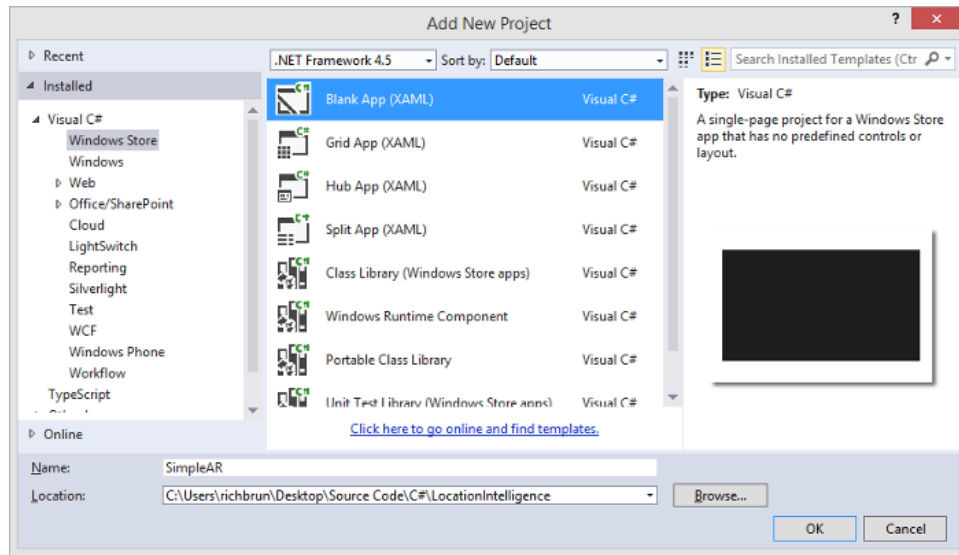
A simple geospatial augmented reality app can be created using nothing more than the location and compass sensors. The location sensor will provide you with the current location of the user and the compass will give you information on where the user is pointing the device. This will work fine in two dimensions, but if you want to take elevations into account, then you will need to look at the inclinometer sensor. The inclinometer measures the angle of rotation around the axes of the device and exposes these angles as yaw, pitch, and roll readings. Now as I mentioned, these will work fine for simple augmented reality apps, but you will likely find that things don't move as smoothly on the screen as you would like. For more advance augmented reality apps you will likely make use of the orientation sensor which combines the accelerometer, compass, and gyrometer (measures angular velocity about the axis) to report even more sensitive movements than any of those sensors can on their own. This makes the movement within the app much smoother, but also makes the mathematics much more complicated as a rotation matrix is returned by this sensor.

Creating an Augmented Reality App

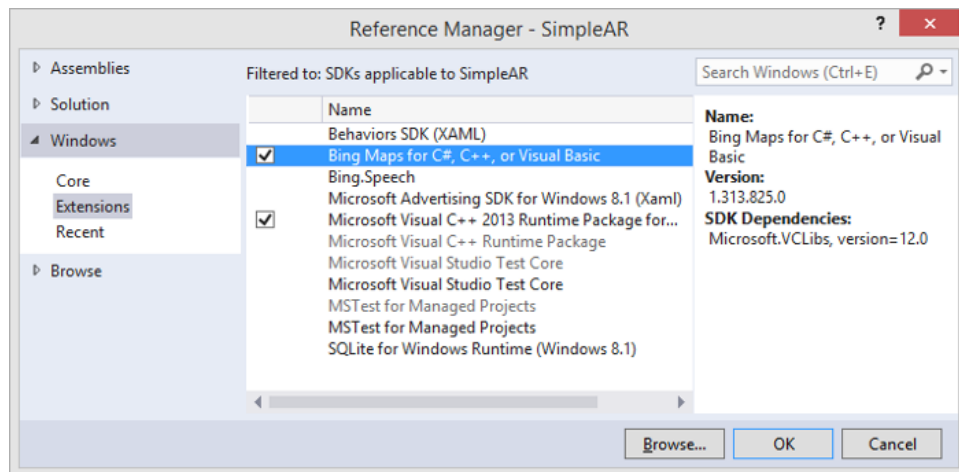
In this chapter we are going to create a simple augmented reality app that shows nearby points of interests. The NAVTEQ point of interest data from the Bing Spatial Data Services will be used as the data source for the app. In addition to making use of the location and compass sensors we will also make use of the simple orientation sensor. The simple orientation sensor will be used to rotate the video from the rear facing camera as the device orientation changes. This sensor will also be used to switch between an augmented reality view and a map view when the device is face up or down.

Creating the Base Project

To get started open up Visual Studios and create a new project in your preferred language; JavaScript, C# or Visual Basic. Select the **Blank App** template and call the application **SimpleAR** and press **OK**.

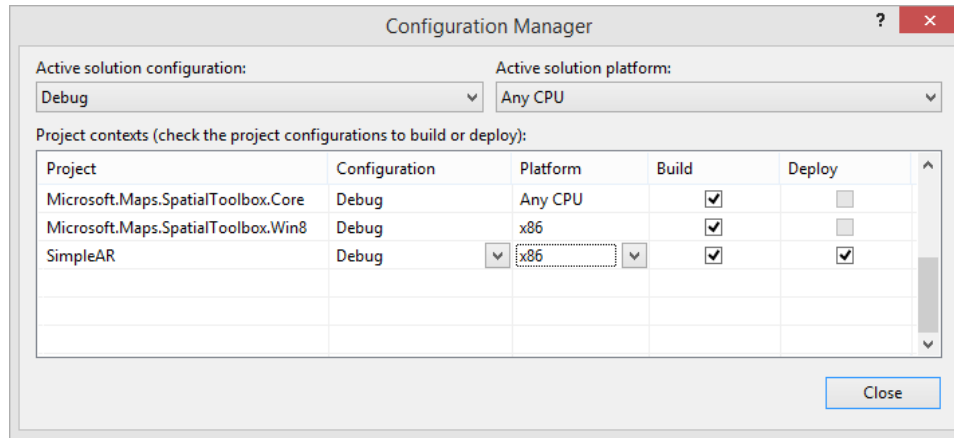


Add a reference to the Bing Maps SDK. To do this, right click on the **References** folder and press **Add Reference**. Select **Windows** -> **Extensions** select **Bing Maps for C#, C++ and Visual Basic** or **Bing Maps for JavaScript**. If you do not see this option ensure that you have installed the Bing Maps SDK for Windows Store apps.



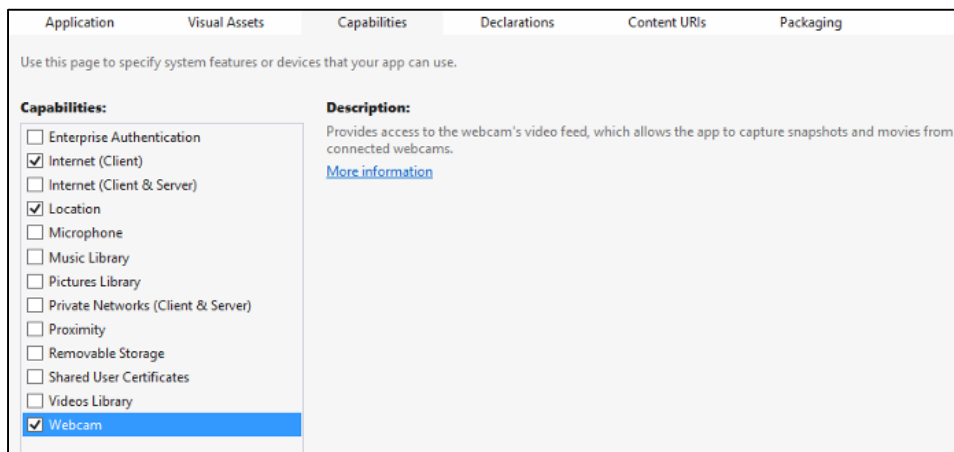
If using C# or Visual Basic, add a reference to the **Microsoft Visual C++ Runtime Package** as this is required by the Bing Maps SDK. Next get a copy of the **SpatialToolbox** library, either copy it from the code samples for this book or download it from the CodePlex site (<http://mapstoolbox.codeplex.com/>). Navigate to the folder where the **SimpleAR** project is located and add a copy of the **Microsoft.Maps.SpatialToolbox.Core** and **Microsoft.Maps.SpatialToolbox.Win8** libraries to that folder. Add these libraries to the app by right click on the solution folder in Visual Studios and selecting **Add** -> **Existing Project**. Next add references to these libraries by right clicking on the **References** folder and pressing **Add Reference**. Select **Solution** -> **Projects**, and then select the **Microsoft.Maps.SpatialToolbox.Core** and **Microsoft.Maps.SpatialToolbox.Win8** projects.

If you notice that there is a little yellow indicator on the references that you just added. The reason for this is that in the C++ runtime package you have to set the **Active solution platform** in Visual Studio to one of the following options; **ARM**, **x86** or **x64**. To do this, right click on the Solution folder and select **Properties**. Then go to **Configuration Properties** -> **Configuration**. Find your project and under the **Platform** column set the target platform to x86. Press OK and the yellow indicator should disappear from our references.



If you are using JavaScript right click on the **js** folder and select **Add -> New Item** and create a new JavaScript file called **SimpleAR.js**. We will put all our JavaScript for this application in this file to keep things clean.

One final step in setting up the base project is to set the capabilities in the package manifest of the app to access the user's location and webcam. To do this, open the **package.appxmanifest** file and select the Capabilities tab at the top. Under the Capabilities section check the Location and Webcam options.



Creating the Application UI

The app will consist of two views; an augmented reality view and a map view. The augmented reality view consists of three parts; a video preview from the rear facing camera, a canvas to render augmented items on, and text which is updated with the current heading reading of the device. The video preview will make use of the HTML5 **video** tag in the JavaScript version of the app. A **CaptureElement** will be used in the C#/Visual Basic version of the app.

If using JavaScript, open the **default.html** file and update the HTML to the following. This will add references to the **SimpleAR.js** and Bing Maps JavaScript files required by the app. It also adds the HTML required to create the augmented reality and map view.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>SimpleAR</title>

  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
```



```

<script src="//Microsoft.WinJS.2.0/js/base.js"></script>
<script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

<!-- SimpleAR references -->
<link href="/css/default.css" rel="stylesheet" />
<script src="/js/default.js"></script>

<!-- Bing Map Control references -->
<script type="text/javascript" src="ms-
appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>

<script src="/js/SimpleAR.js"></script>
</head>
<body>
  <div id="ARView">
    <video id="PreviewScreen"></video>
    <canvas id="ItemCanvas"></canvas>
    <span id="CompassReading"></span>
  </div>
  <div id="MapView">
    <div id="MyMap"></div>
  </div>
</body>
</html>

```

Next open the **default.css** file and update it with the following CSS styles. These styles are used to make both views overlap each other and fill the entire screen. The **.gpsPin** style will be used later to apply a rotation transform to a pushpin that displays the users location on the map.

```

#ARView, #PreviewScreen, #ItemCanvas, #MapView, #MyMap {
  position:absolute;
  top:0px;
  left:0px;
  width:100%;
  height:100%;
}

#CompassReading {
  position:absolute;
  top:0px;
  width:100%;
  text-align:center;
  font-size:24px;
}

.gpsPin {
  transform-origin:center 75%;
}

```

If you are using C# or Visual Basic open the **MainPage.xaml** file and update it with the following XAML.

```

<Page
  x:Class="SimpleAR.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:SimpleAR"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:m="using:Bing.Maps">

```

```

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid Name="ARView">
        <CaptureElement Name="PreviewScreen" Stretch="UniformToFill"/>
        <Canvas Name="ItemCanvas"/>
        <TextBlock Name="CompassReading" FontSize="24" HorizontalAlignment="Center"
VerticalAlignment="Top"/>
    </Grid>
    <Grid Name="MapView">
        <m:Map Name="MyMap" Credentials="YOUR_BING_MAPS_KEY"/>
    </Grid>
</Grid>
</Page>

```

In this app we will display the user's location on the map using a custom pushpin which indicates the direction in which the device is pointing. Add the following image to the **images** or **Assets** folder in your project.



SightPin.png

This image will be used as the custom pushpin and will be rotated using a rotation transform. The standard pushpin class in the JavaScript Bing Maps control will be used to do this and implemented later in this chapter. If you are using C# or Visual Basic we will create a user control to use as the custom pushpin. To do this right click on the project and select **Add -> New Item** and create a new user control called **UserPushpin.xaml**. Open the **UserPushpin.xaml** file and update it with the following XAML.

```

<UserControl
    x:Class="SimpleAR.UserPushpin"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:SimpleAR"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:m="using:Bing.Maps">

    <StackPanel>
        <Image Source="/Assets/SightPin.png" Height="40" Width="40" Margin="-20,-30,0,0">
            <Image.RenderTransform>
                <RotateTransform CenterX="20" CenterY="30" Angle="{Binding}"/>
            </Image.RenderTransform>
        </Image>
    </StackPanel>
</UserControl>

```

This user control will allow us to rotate the sight pin image by passing an angle between 0 and 360 to the **DataContext** property of the user control.

Initializing the App

When the application loads we will need to load the map, retrieve a session key from it, add an layer for displaying the points of interest on, and add a pushpin for displaying the user's location. We will also want to define a number of variables that we will use throughout the app. To do this open the **SimpleAR.js** or the **MainPage.xaml.cs** file and update it with the following code.

JavaScript

```

(function () {
    var app = WinJS.Application;

    var PreviewScreen, MapView, CompassReading, MyMap;
    var mediaCapture, orientationSensor, compass, gps;

    var earthRadiusKM = 6378.135;
    var movementThreshold = 100;
    var defaultSearchRadius = 1;

    var map, sessionKey, pinLayer, gpsPin;
    var currentHeading = 0, currentLocation, poiLocations;

    app.onloaded = function (args) {
    };

    app.onunload = function (args) {
    };

    function loadMap() {
        map = new Microsoft.Maps.Map(MyMap, {
            credentials: "YOUR_BING_MAPS_KEY"
        });

        map.getCredentials(function (c) {
            sessionKey = c;
        });

        pinLayer = new Microsoft.Maps.EntityCollection();
        map.entities.push(pinLayer);

        gpsPin = new Microsoft.Maps.Pushpin(map.getCenter(), {
            icon: "images/SightPin.png",
            typeName: "gpsPin",
            width: 40,
            height: 40
        });
        map.entities.push(gpsPin);
    }
})();

```

C#

```

using Bing.Maps;
using Microsoft.Maps.SpatialToolbox;
using Microsoft.Maps.SpatialToolbox.Bing.NavteqPoiSchema;
using System;
using System.Linq;
using System.Runtime.Serialization.Json;
using System.Threading.Tasks;
using Windows.Devices.Enumeration;
using Windows.Devices.Geolocation;
using Windows.Devices.Sensors;
using Windows.Media.Capture;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;

```

```

namespace SimpleAR
{
    public sealed partial class MainPage : Page
    {
        private SimpleOrientationSensor orientationSensor;
        private Compass compass;
        private Geolocator gps;

        private uint movementThreshold = 100;
        private double defaultSearchRadius = 1;

        private string sessionKey;
        private MapLayer pinLayer;
        private UserPushpin gpsPin;

        private Location currentLocation = null;
        private double currentHeading = 0;
        private Result[] poiLocations = null;

        public MainPage()
        {
            this.InitializeComponent();

            MyMap.Loaded += async (s, e) =>
            {
                try
                {
                    sessionKey = await MyMap.GetSessionIdAsync();

                    pinLayer = new MapLayer();
                    MyMap.Children.Add(pinLayer);

                    gpsPin = new UserPushpin();
                    MyMap.Children.Add(gpsPin);
                }
                catch { }
            };
        }
    }
}

```

Visual Basic

```

Imports Windows.Devices.Sensors
Imports Windows.Devices.Geolocation
Imports Bing.Maps
Imports Microsoft.Maps.SpatialToolbox.Bing.NavteqPoiSchema
Imports Windows.Media.Capture
Imports Windows.Devices.Enumeration
Imports System.Runtime.Serialization.Json
Imports Microsoft.Maps.SpatialToolbox

Public NotInheritable Class MainPage
    Inherits Page

    Private orientationSensor As SimpleOrientationSensor
    Private compass As Compass
    Private gps As Geolocator

    Private movementThreshold As UInteger = 100

```

```

Private defaultSearchRadius As Double = 1

Private sessionKey As String
Private pinLayer As MapLayer
Private gpsPin As UserPushpin

Private currentLocation As Location = Nothing
Private currentHeading As Double = 0
Private poiLocations As Result() = Nothing

Public Sub New()
    Me.InitializeComponent()

    AddHandler MyMap.Loaded,
        Async Sub()
            Try
                sessionKey = Await MyMap.GetSessionIdAsync()

                pinLayer = New MapLayer()
                MyMap.Children.Add(pinLayer)

                gpsPin = New UserPushpin()
                MyMap.Children.Add(gpsPin)
            Catch
            End Try
        End Sub
    End Sub
End Class

```

Adding Helper Methods

This app has a lot of moving parts in it. To make things a bit easier we will create a number of helper methods. These methods will help us find the rear facing camera, start and stop the camera, and perform a search for nearby NAVTEQ point of interest. In JavaScript we will also include methods that calculate distances and headings between coordinates. Add the following methods to the **SimpleAR.js** or the **MainPage.xaml.cs** file.

JavaScript

```

function startCamera() {
    PreviewScreen = document.getElementById("PreviewScreen");
    PreviewScreen.msZoom = true;

    findRearFacingCamera().then(function (cameraId) {
        try {
            //Use a specific camera
            if (cameraId != null && cameraId != "") {
                var settings = new
Windows.Media.Capture.MediaCaptureInitializationSettings();
                settings.videoDeviceId = cameraId;
                settings.streamingCaptureMode =
Windows.Media.Capture.StreamingCaptureMode.video;

                mediaCapture = new Windows.Media.Capture.MediaCapture();
                mediaCapture.initializeAsync(settings).then(function () {
                    PreviewScreen.src = URL.createObjectURL(mediaCapture);
                    PreviewScreen.play();
                });
            }
        }
    })
}

```

```

        catch(e){ }
    });
}

function stopCamera()
{
    if (PreviewScreen)
    {
        //Stop the camera if the video source is not null
        PreviewScreen.pause();
        PreviewScreen.src = null;
    }
}

function findRearFacingCamera()
{
    return new WinJS.Promise(function (complete) {
        //Get all video capture devices

        Windows.Devices.Enumeration.DeviceInformation.findAllAsync(Windows.Devices.Enumeration.DeviceClass.videoCapture).then(function(args){
            var device = null;

            //First look using panel, this is the best approach
            for(var i = 0; i < args.length; i++){
                if (args[i].enclosureLocation &&
                    args[i].enclosureLocation.panel ==
Windows.Devices.Enumeration.Panel.back){
                    device = args[i];
                    break;
                }
            }

            //If camera wasn't found using panel location, look for a device that has "back"
            or "rear" in the name or ID
            if(device == null){
                for(var i = 0; i < args.length; i++){
                    if(args[i].name.toLowerCase().indexOf("back") > 0 ||
                        args[i].id.toLowerCase().indexOf("back") > 0 ||
                        args[i].name.toLowerCase().indexOf("rear") > 0 ||
                        args[i].id.toLowerCase().indexOf("rear") > 0)
                    {
                        device = args[i];
                        break;
                    }
                }
            }

            //If a device was found return it's ID
            if (device)
            {
                complete(device.id);
            }

            complete(null);
        });
    });
}

function navteqPoiSearch(center){
    return new WinJS.Promise(function (complete) {

```

```

        var baseUrl;

        //Switch between the NAVTEQ POI data sets for NA and EU based on the longitude
value.
        if (center.longitude < -30) {
            baseUrl =
"http://spatial.virtualearth.net/REST/v1/data/f22876ec257b474b82fe2ffcb8393150/NavteqNA/Navt
eqPOIs";
        }
        else {
            baseUrl =
"http://spatial.virtualearth.net/REST/v1/data/c2ae584bbccc4916a0acf75d1e6947b4/NavteqEU/Navt
eqPOIs";
        }

        var query = baseUrl + "?spatialFilter=nearby(" + center.latitude + "," +
center.longitude +
            "," + defaultSearchRadius + ")&$format=json&key=" + sessionKey;

        //Get response from NAVTEQ POI data source
WinJS.xhr({ url: query }).then(function (e) {
    //Parse the text response into JSON
    var r = JSON.parse(e.responseText);

    complete(r);
});
});
}

function calculateHeading(start, end){
    var dLat1 = (Math.PI / 180) * start.latitude;
    var dLat2 = (Math.PI / 180) * end.latitude;
    var dLon = (Math.PI / 180) * (end.longitude - start.longitude);
    var y = Math.sin(dLon) * Math.cos(dLat2);
    var x = Math.cos(dLat1) * Math.sin(dLat2) - Math.sin(dLat1) * Math.cos(dLat2) *
Math.cos(dLon);
    return ((180 / Math.PI) * Math.atan2(y, x) + 360) % 360;
}

function haversineDistance(start, end, earthRadius){
    var dLat = (Math.PI / 180) * (end.latitude - start.latitude);
    var dLon = (Math.PI / 180) * (end.longitude - start.longitude);

    var a = Math.sin(dLat / 2) * Math.sin(dLat / 2) + Math.cos((Math.PI / 180) *
(start.latitude)) *
        Math.cos((Math.PI / 180) * (start.latitude)) * Math.sin(dLon / 2) * Math.sin(dLon /
2);
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

    return earthRadius * c;
}

```

C#

```

private async Task StartCamera()
{
    var source = new Windows.Media.Capture.MediaCapture();
    try
    {
        var cameraId = await FindRearFacingCamera();
    }
}

```

```

        //Use a specific camera
        if (!string.IsNullOrEmpty(cameraId))
        {
            var settings = new MediaCaptureInitializationSettings();
            settings.VideoDeviceId = cameraId;
            settings.StreamingCaptureMode = StreamingCaptureMode.Video;

            await source.InitializeAsync(settings);

            PreviewScreen.Source = source;

            //Start video preview
            await source.StartPreviewAsync();
        }
    }
    catch { }
}

private async void StopCamera()
{
    try
    {
        if (PreviewScreen.Source != null)
        {
            //Stop the camera if the video source is not null
            await PreviewScreen.Source.StopPreviewAsync();
            PreviewScreen.Source = null;
        }
    }
    catch { }
}

private async Task<string> FindRearFacingCamera()
{
    //Get all video capture devices
    var devices = await DeviceInformation.FindAllAsync(DeviceClass.VideoCapture);

    //First look using panel, this is the best approach
    var device = (from d in devices
        where d.EnclosureLocation != null &&
        d.EnclosureLocation.Panel == Windows.Devices.Enumeration.Panel.Back
        select d).FirstOrDefault();

    //If camera wasn't found using panel location, look for a device that has "back" or
    "rear" in the name or ID
    if (device == null)
    {
        device = (from d in devices
            where d.Name.ToLower().Contains("back") ||
            d.Id.ToLower().Contains("back") ||
            d.Name.ToLower().Contains("rear") ||
            d.Id.ToLower().Contains("rear")
            select d).FirstOrDefault();
    }

    //If a device was found return it's ID
    if (device != null)
    {
        return device.Id;
    }
}

```



```

        return null;
    }

    private async Task<Response> NavteqPoiSearch(Location center)
    {
        string baseUrl;

        //Switch between the NAVTEQ POI data sets for NA and EU based on the longitude value.
        if (center.Longitude < -30)
        {
            baseUrl =
                "http://spatial.virtualearth.net/REST/v1/data/f22876ec257b474b82fe2ffcb8393150/NavteqNA/NavteqPOIs";
        }
        else
        {
            baseUrl =
                "http://spatial.virtualearth.net/REST/v1/data/c2ae584bbccc4916a0acf75d1e6947b4/NavteqEU/NavteqPOIs";
        }

        string query =
            string.Format("{0}?spatialFilter=nearby({1:N5},{2:N5},{3})&$format=json&key={4}",
                baseUrl, center.Latitude, center.Longitude, defaultSearchRadius, sessionKey);

        return await GetResponse<Response>(new Uri(query));
    }

    private async Task<T> GetResponse<T>(Uri uri)
    {
        System.Net.Http.HttpClient client = new System.Net.Http.HttpClient();
        var response = await client.GetAsync(uri);

        using (var stream = await response.Content.ReadAsStreamAsync())
        {
            DataContractJsonSerializer ser = new DataContractJsonSerializer(typeof(T));
            return (T)ser.ReadObject(stream);
        }
    }
}

```

Visual Basic

```

Private Async Function StartCamera() As Task
    Dim source = New Windows.Media.Capture.MediaCapture()
    Try
        Dim cameraId = Await FindRearFacingCamera()

        'Use a specific camera
        If Not String.IsNullOrEmpty(cameraId) Then
            Dim settings = New MediaCaptureInitializationSettings()
            settings.VideoDeviceId = cameraId
            settings.StreamingCaptureMode = StreamingCaptureMode.Video

            Await source.InitializeAsync(settings)

            PreviewScreen.Source = source

            'Start video preview
            Await source.StartPreviewAsync()
        End If
    Catch ex As Exception
        'Handle exception
    End Try
}

```

```

        End If
    Catch
    End Try
End Function

Private Async Sub StopCamera()
    Try
        If PreviewScreen.Source IsNot Nothing Then
            'Stop the camera if the video source is not null
            Await PreviewScreen.Source.StopPreviewAsync()
            PreviewScreen.Source = Nothing
        End If
    Catch
    End Try
End Sub

Private Async Function FindRearFacingCamera() As Task(Of String)
    'Get all video capture devices
    Dim devices = Await DeviceInformation.FindAllAsync(DeviceClass.VideoCapture)

    'First look using panel, this is the best approach
    Dim device = (From d In devices Where d.EnclosureLocation IsNot Nothing AndAlso
d.EnclosureLocation.Panel = Windows.Devices.Enumeration.Panel.Back).FirstOrDefault()

    'If camera wasn't found using panel location, look for a device that has "back" or
"rear" in the name or ID
    If device Is Nothing Then
        device = (From d In devices Where d.Name.ToLower().Contains("back") OrElse
d.Id.ToLower().Contains("back") OrElse d.Name.ToLower().Contains("rear") OrElse
d.Id.ToLower().Contains("rear")).FirstOrDefault()
    End If

    'If a device was found return it's ID
    If device IsNot Nothing Then
        Return device.Id
    End If

    Return Nothing
End Function

Private Async Function NavteqPoiSearch(center As Location) As Task(Of Response)
    Dim baseUrl As String

    'Switch between the NAVTEQ POI data sets for NA and EU based on the longitude value.
    If center.Longitude < -30 Then
        baseUrl =
"http://spatial.virtualearth.net/REST/v1/data/f22876ec257b474b82fe2ffcb8393150/NavteqNA/Navt
eqPOIs"
    Else
        baseUrl =
"http://spatial.virtualearth.net/REST/v1/data/c2ae584bbccc4916a0acf75d1e6947b4/NavteqEU/Navt
eqPOIs"
    End If

    Dim query As String =
String.Format("{0}?spatialFilter=nearby({1:N5},{2:N5},{3})&$format=json&key={4}", baseUrl,
center.Latitude, center.Longitude, defaultSearchRadius, sessionKey)

    Return Await GetResponse(Of Response)(New Uri(query))
End Function

```

```

Private Async Function GetResponse(Of T)(uri As Uri) As Task(Of T)
    Dim client As New System.Net.Http.HttpClient()
    Dim response = Await client.GetAsync(uri)

    Using stream = Await response.Content.ReadAsStreamAsync()
        Dim ser As New DataContractJsonSerializer(GetType(T))
        Return DirectCast(ser.ReadObject(stream), T)
    End Using
End Function

```

Rendering Items in the Augmented Reality View

Before diving into the sensors and wiring up the app to find locations we will create a method that will calculate the heading of all the nearby points of interest and their distance from the user. This method will then use this information to position the item on the canvas above the video from the rear facing camera. In this method we need to perform calculations to determine which locations are in view and where they should be placed on the screen. To do this we need to define a field of view. A field of view is an angle which we use to define the area that is in view. I've chosen a field of view of 45 degrees. This means that items that having a heading that are within 22.5 degrees to the left or right of the direction in which we point the device will be displayed on the screen. To represent each item on the canvas we could create a custom control that has a number of additional features, but to keep things simple we will just display the name of the point of interest. We will also place the items on the screen such that the closer items appear near the bottom of the screen and the further items appear near the top of the screen. Add the following method to the **SimpleAR.js** or the **MainPage.xaml.cs** file.

JavaScript

```

function updateARView() {
    if (currentLocation)
    {
        //Clear the canvas by setting its size.
        ItemCanvas.width = window.innerWidth;
        ItemCanvas.height = window.innerHeight;

        var context = ItemCanvas.getContext("2d");

        if (poiLocations && poiLocations.length > 0 && context){
            context.font = "24px Arial";
            context.fillStyle = "#ffffff";
            context.textAlign = "center";

            for(var i = 0; i < poiLocations.length; i++){
                var c = new Microsoft.Maps.Location(poiLocations[i].Latitude,
                poiLocations[i].Longitude);
                var poiHeading = calculateHeading(currentLocation, c);
                var diff = currentHeading - poiHeading;

                if(diff > 180){
                    diff = currentHeading - (poiHeading + 360);
                }
                else if (diff < -180)
                {
                    diff = currentHeading + 360 - poiHeading;
                }

                if(Math.abs(diff) <= 22.5)
                {
                    var distance = haversineDistance(currentLocation, c, earthRadiusKM);

```

```

        var left = 0;

        if(diff > 0){
            left = ItemCanvas.width / 2 * ((22.5 - diff) / 22.5);
        }
        else
        {
            left = ItemCanvas.width / 2 * (1 + -diff / 22.5);
        }

        var top = ItemCanvas.height * (1 - distance / defaultSearchRadius);

        context.fillText(poiLocations[i].Name, left, top);
    }
}
}
}
}
}

```

C#

```

private void UpdateARView()
{
    if (currentLocation != null)
    {
        ItemCanvas.Children.Clear();

        if (poiLocations != null && poiLocations.Length > 0)
        {
            var center = new Coordinate(currentLocation.Latitude,
currentLocation.Longitude);

            foreach (var poi in poiLocations)
            {
                var c = new Coordinate(poi.Latitude, poi.Longitude);
                var poiHeading = SpatialTools.CalculateHeading(center, c);
                var diff = currentHeading - poiHeading;

                if (diff > 180)
                {
                    diff = currentHeading - (poiHeading + 360);
                }
                else if (diff < -180)
                {
                    diff = currentHeading + 360 - poiHeading;
                }

                if (Math.Abs(diff) <= 22.5)
                {
                    var distance = SpatialTools.HaversineDistance(center, c,
DistanceUnits.KM);

                    double left = 0;

                    if (diff > 0)
                    {
                        left = ItemCanvas.ActualWidth / 2 * ((22.5 - diff) / 22.5);
                    }
                    else
                    {

```

```

        left = ItemCanvas.ActualWidth / 2 * (1 + -diff / 22.5);
    }

    double top = ItemCanvas.ActualHeight * (1 - distance /
defaultSearchRadius);

    var tb = new TextBlock()
    {
        Text = poi.Name,
        FontSize = 24,
        TextAlignment = Windows.UI.Xaml.TextAlignment.Center,
        HorizontalAlignment = Windows.UI.Xaml.HorizontalAlignment.Center
    };

    Canvas.SetLeft(tb, left);
    Canvas.SetTop(tb, top);
    ItemCanvas.Children.Add(tb);
}
}
}
}
}
}
}
}
}
}

```

Visual Basic

```

Private Sub UpdateARView()
    If currentLocation IsNot Nothing Then
        ItemCanvas.Children.Clear()

        If poiLocations IsNot Nothing AndAlso poiLocations.Length > 0 Then
            Dim center = New Coordinate(currentLocation.Latitude,
currentLocation.Longitude)

            For Each poi As Result In poiLocations
                Dim c = New Coordinate(poi.Latitude, poi.Longitude)
                Dim poiHeading = SpatialTools.CalculateHeading(center, c)
                Dim diff = currentHeading - poiHeading

                If diff > 180 Then
                    diff = currentHeading - (poiHeading + 360)
                ElseIf diff < -180 Then
                    diff = currentHeading + 360 - poiHeading
                End If

                If Math.Abs(diff) <= 22.5 Then
                    Dim distance = SpatialTools.HaversineDistance(center, c,
DistanceUnits.KM)

                    Dim left As Double = 0

                    If diff > 0 Then
                        left = ItemCanvas.ActualWidth / 2 * ((22.5 - diff) / 22.5)
                    Else
                        left = ItemCanvas.ActualWidth / 2 * (1 + -diff / 22.5)
                    End If

                    Dim top As Double = ItemCanvas.ActualHeight * (1 - distance /
defaultSearchRadius)

                    Dim tb = New TextBlock()

```

```

        tb.Text = poi.Name
        tb.FontSize = 24
        tb.TextAlignment = Windows.UI.Xaml.TextAlignment.Center
        tb.HorizontalAlignment = Windows.UI.Xaml.HorizontalAlignment.Center

        Canvas.SetLeft(tb, left)
        Canvas.SetTop(tb, top)
        ItemCanvas.Children.Add(tb)
    End If
Next
End If
End If
End Sub

```

Handling App Navigation

When the user navigates to the app we will want to start the camera and attach all the required events to the sensors. When the user navigates away from the app we will want to stop the camera and detach these events. Add the following code to the **SimpleAR.js** or the **MainPage.xaml.cs** file to handle this navigation. Note that the event handlers will be added in the next section of this chapter.

JavaScript

```

app.onloaded = function (args) {
    //Get DOM references
    ItemCanvas = document.getElementById("ItemCanvas");
    CompassReading = document.getElementById("CompassReading");
    MapView = document.getElementById("MapView");
    MyMap = document.getElementById("MyMap");

    //Load the Map
    Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: loadMap });

    //Start the camera
    startCamera();

    window.onfocus = startCamera;
    window.onblur = stopCamera;

    orientationSensor = Windows.Devices.Sensors.SimpleOrientationSensor.getDefault();
    if (orientationSensor) {
        orientationSensor.addEventListener("orientationchanged",
        simpleOrientationSensorReadingChanged);
        updateOrientation(orientationSensor.getCurrentOrientation());
    }

    compass = Windows.Devices.Sensors.Compass.getDefault();
    if (compass) {
        compass.reportInterval = compass.minimumReportInterval > 16 ?
            compass.minimumReportInterval : 16;

        compass.addEventListener("readingchanged", compassReadingChanged);
        compassChanged(compass.getCurrentReading());
    }

    gps = new Windows.Devices.Geolocation.Geolocator();
    gps.movementThreshold = movementThreshold;
    gps.addEventListener("positionchanged", gpsPositionChanged);
}

```

```

    if (gps.locationStatus == Windows.Devices.Geolocation.PositionStatus.ready) {
        gps.getGeopositionAsync().then(function (pos) {
            if (pos != null && pos.coordinate != null && pos.coordinate.point != null) {
                gpsChanged(pos.coordinate.point.position);
            }
        });
    }
};

app.onunload = function (args) {
    if (orientationSensor) {
        orientationSensor.removeEventListener("orientationchanged",
        simpleOrientationSensorReadingChanged);
    }

    if (compass) {
        compass.removeEventListener("readingchanged", compassReadingChanged);
        compass.reportInterval = 0;
    }

    geolocator.removeEventListener("positionchanged", gpsPositionChanged);
};

```

C#

```

protected async override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    try
    {
        await StartCamera();

        orientationSensor = SimpleOrientationSensor.GetDefault();
        if (orientationSensor != null)
        {
            orientationSensor.OrientationChanged +=
            SimpleOrientationSensorReadingChanged;
            UpdateOrientation(orientationSensor.GetCurrentOrientation());
        }

        compass = Compass.GetDefault();
        if (compass != null)
        {
            compass.ReadingChanged += CompassReadingChanged;
            CompassChanged(compass.GetCurrentReading());
        }

        gps = new Geolocator();
        gps.MovementThreshold = movementThreshold;
        gps.PositionChanged += GpsPositionChanged;

        if (gps.LocationStatus == PositionStatus.Ready)
        {
            var pos = await gps.GetGeopositionAsync();
            if (pos != null && pos.Coordinate != null && pos.Coordinate.Point != null)
            {
                GpsChanged(pos.Coordinate.Point.Position);
            }
        }
    }
}

```

```

    }
    catch { }
}

```

Visual Basic

```

Protected Overrides Async Sub OnNavigatedTo(e As NavigationEventArgs)
    MyBase.OnNavigatedTo(e)

    Try
        Await StartCamera()

        orientationSensor = SimpleOrientationSensor.GetDefault()
        If orientationSensor IsNot Nothing Then
            AddHandler orientationSensor.OrientationChanged, AddressOf
SimpleOrientationSensorReadingChanged
            UpdateOrientation(orientationSensor.GetCurrentOrientation())
        End If

        compass = compass.GetDefault()
        If compass IsNot Nothing Then
            AddHandler compass.ReadingChanged, AddressOf CompassReadingChanged
            CompassChanged(compass.GetCurrentReading())
        End If

        gps = New Geolocator()
        gps.MovementThreshold = movementThreshold
        AddHandler gps.PositionChanged, AddressOf GpsPositionChanged

        If gps.LocationStatus = PositionStatus.Ready Then
            Dim pos = Await gps.GetGeopositionAsync()
            If pos IsNot Nothing AndAlso pos.Coordinate IsNot Nothing AndAlso
pos.Coordinate.Point IsNot Nothing Then
                GpsChanged(pos.Coordinate.Point.Position)
            End If
        End If
    Catch
    End Try
End Sub

```

Adding the Sensor Event Handlers

The first sensor that we will add the event handler for is the simple orientation sensor. We will use this sensor to do two jobs. The first is to rotate the video as the device is rotated. This will ensure that the video is properly positioned and is in line with what the device is pointing at. The second job is to switch to the map view when the device orientation is face up or down. Add the following code for the simple orientation sensor event handler to the **SimpleAR.js** or the **MainPage.xaml.cs** file.

JavaScript

```

function simpleOrientationSensorReadingChanged(e) {
    updateOrientation(e.orientation);
}

function updateOrientation(orientation) {
    var videoRotation = Windows.Media.Capture.VideoRotation.none;
    var showMap = false;

```



```

switch (orientation) {
    case Windows.Devices.Sensors.SimpleOrientation.notRotated:
        videoRotation = Windows.Media.Capture.VideoRotation.none;
        break;
    case Windows.Devices.Sensors.SimpleOrientation.rotated90DegreesCounterclockwise:
        videoRotation = Windows.Media.Capture.VideoRotation.clockwise90Degrees;
        break;
    case
Windows.Devices.Sensors.SimpleOrientation.rotated180DegreesCounterclockwise:
        videoRotation = Windows.Media.Capture.VideoRotation.clockwise180Degrees;
        break;
    case
Windows.Devices.Sensors.SimpleOrientation.rotated270DegreesCounterclockwise:
        videoRotation = Windows.Media.Capture.VideoRotation.clockwise270Degrees;
        break;
    case Windows.Devices.Sensors.SimpleOrientation.facedown:
    case Windows.Devices.Sensors.SimpleOrientation.faceup:
        showMap = true;
        break;
    default:
        break;
}

if (mediaCapture) {
    try{
        mediaCapture.setPreviewRotation(videoRotation);
    }catch(e){}
}

if (showMap) {
    MapView.style.display = "";
}
else {
    MapView.style.display = "none";
}
}

```

C#

```

private void SimpleOrientationSensorReadingChanged(SimpleOrientationSensor sender,
SimpleOrientationSensorOrientationChangedEventArgs args)
{
    UpdateOrientation(args.Orientation);
}

private async void UpdateOrientation(SimpleOrientation orientation)
{
    try
    {
        VideoRotation videoRotation = VideoRotation.None;
        bool showMap = false;

        switch (orientation)
        {
            case SimpleOrientation.NotRotated:
                videoRotation = VideoRotation.None;
                break;
            case SimpleOrientation.Rotated90DegreesCounterclockwise:
                videoRotation = VideoRotation.Clockwise90Degrees;
                break;

```

```

        case SimpleOrientation.Rotated180DegreesCounterclockwise:
            videoRotation = VideoRotation.Clockwise180Degrees;
            break;
        case SimpleOrientation.Rotated270DegreesCounterclockwise:
            videoRotation = VideoRotation.Clockwise270Degrees;
            break;
        case SimpleOrientation.Facedown:
        case SimpleOrientation.Faceup:
            showMap = true;
            break;
        default:
            break;
    }

    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
    {
        if (PreviewScreen.Source != null)
        {
            PreviewScreen.Source.SetPreviewRotation(videoRotation);
        }

        if (showMap)
        {
            MapView.Visibility = Windows.UI.Xaml.Visibility.Visible;
        }
        else
        {
            MapView.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
        }
    });
}
catch { }
}

```

Visual Basic

```

Private Sub SimpleOrientationSensorReadingChanged(sender As SimpleOrientationSensor,
args As SimpleOrientationSensorOrientationChangedEventArgs)
    UpdateOrientation(args.Orientation)
End Sub

Private Async Sub UpdateOrientation(orientation As SimpleOrientation)
    Dim videoRotation As VideoRotation = videoRotation.None
    Dim showMap As Boolean = False

    Select Case orientation
        Case SimpleOrientation.NotRotated
            videoRotation = videoRotation.None
            Exit Select
        Case SimpleOrientation.Rotated90DegreesCounterclockwise
            videoRotation = videoRotation.Clockwise90Degrees
            Exit Select
        Case SimpleOrientation.Rotated180DegreesCounterclockwise
            videoRotation = videoRotation.Clockwise180Degrees
            Exit Select
        Case SimpleOrientation.Rotated270DegreesCounterclockwise
            videoRotation = videoRotation.Clockwise270Degrees
            Exit Select
        Case SimpleOrientation.Facedown, SimpleOrientation.Faceup
            showMap = True
    End Select

```

```

        Exit Select
    Case Else
        Exit Select
    End Select

    Await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
        Sub()
            If PreviewScreen.Source IsNot Nothing Then
                PreviewScreen.Source.SetPreviewRotation(videoRotation)
            End If

            If showMap Then
                MapView.Visibility = Windows.UI.Xaml.Visibility.Visible
            Else
                MapView.Visibility = Windows.UI.Xaml.Visibility.Collapsed
            End If
        End Sub)
End Sub

```

The second sensor that we will add the event handler for is the compass. When the compass heading changes the position of all the items in the augmented reality view will be updated. The pushpin that displays the user's position on the map will also be rotated to align with the compass heading. Add the following code for the compass event handler to the **SimpleAR.js** or the **MainPage.xaml.cs** file.

JavaScript

```

function compassReadingChanged(e) {
    var reading = e.reading;
    compassChanged(reading);
}

function compassChanged(reading) {
    CompassReading.innerHTML = reading.headingMagneticNorth + "";
    currentHeading = reading.headingMagneticNorth;

    //Set GPS pin heading
    var pins = document.getElementsByClassName("gpsPin");
    if (pins.length > 0) {
        var transform = "rotate(" + currentHeading + "deg)";
        for (var i = 0; i < pins.length; i++) {
            pins[i].style.transform = transform;
        }
    }

    //Update locations in view
    updateARView();
}

```

C#

```

private void CompassReadingChanged(Compass sender, CompassReadingChangedEventArgs args)
{
    var reading = sender.GetCurrentReading();
    CompassChanged(reading);
}

private async void CompassChanged(CompassReading reading)
{

```

```

try
{
    await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
    {
        CompassReading.Text = reading.HeadingMagneticNorth + "";
        currentHeading = reading.HeadingMagneticNorth;

        //Set GPS pin heading
        if (gpsPin != null)
        {
            gpsPin.DataContext = currentHeading;
        }

        //Update locations in view
        UpdateARView();
    });
}
catch { }
}

```

Visual Basic

```

Private Sub CompassReadingChanged(sender As Compass, args As
CompassReadingChangedEventArgs)
    Dim reading = sender.GetCurrentReading()
    CompassChanged(reading)
End Sub

Private Async Sub CompassChanged(reading As CompassReading)
    Await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
        Sub()
            CompassReading.Text = CStr(reading.HeadingMagneticNorth)
            currentHeading = reading.HeadingMagneticNorth

            'Set GPS pin heading
            If gpsPin IsNot Nothing Then
                gpsPin.DataContext = currentHeading
            End If

            'Update locations in view
            UpdateARView()
        End Sub)
End Sub

```

The third sensor that we will add the event handler for is the location sensor. When the user's position changes by more than 100 meters a request for points of interest that are within 1KM of the users position will be retrieved from the Bing Spatial Data Services. The returned points of interest will be displayed on the map and the augmented reality view will be updated accordingly. The pushpin representing the user's location will also be updated and the map centered on the users position. Add the following code for the location sensor event handler to the **SimpleAR.js** or the **MainPage.xaml.cs** file.

JavaScript

```

function gpsPositionChanged(args) {
    if (args.position != null && args.position.coordinate != null &&
args.position.coordinate.point != null) {
        gpsChanged(args.position.coordinate.point.position);
    }
}

```

```

}

function gpsChanged(position) {
    currentLocation = new Microsoft.Maps.Location(position.latitude,
position.longitude);
    poiLocations = null;

    //Update Location Data
    if (pinLayer && sessionKey) {
        pinLayer.clear();

        navteqPoiSearch(currentLocation).then(function (poi) {
            if (poi != null && poi.d != null &&
                poi.d.results != null &&
                poi.d.results.length > 0) {
                poiLocations = poi.d.results;

                for (var i = 0; i < poiLocations.length; i++) {
                    var loc = new Microsoft.Maps.Location(poiLocations[i].Latitude,
poiLocations[i].Longitude);
                    var pin = new Microsoft.Maps.Pushpin(loc);
                    pinLayer.push(pin);
                }

                //Center on users location
                map.setView({ center: currentLocation, zoom: 16 });
            }

            //Set GPS pin location
            gpsPin.setLocation(currentLocation);

            //Update locations in view
            updateARView();
        });
    }
}

```

C#

```

private void GpsPositionChanged(Geolocator sender, PositionChangedEventArgs args)
{
    if (args.Position != null && args.Position.Coordinate != null &&
args.Position.Coordinate.Point != null)
    {
        GpsChanged(args.Position.Coordinate.Point.Position);
    }
}

private async void GpsChanged(BasicGeoposition position)
{
    try
    {
        await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, async
() =>
        {
            try
            {
                currentLocation = new Location(position.Latitude, position.Longitude);
                poiLocations = null;
            }

```

```

//Update Location Data
if (pinLayer != null)
{
    pinLayer.Children.Clear();

    var poi = await NavteqPoiSearch(currentLocation);
    if (poi != null && poi.ResultSet != null &&
        poi.ResultSet.Results != null &&
        poi.ResultSet.Results.Length > 0)
    {
        poiLocations = poi.ResultSet.Results;

        foreach (var r in poiLocations)
        {
            var loc = new Location(r.Latitude, r.Longitude);

            var pin = new Pushpin();
            pin.Tag = r;
            MapLayer.SetPosition(pin, loc);
            pinLayer.Children.Add(pin);
        }

        //Center on users location
        MyMap.SetView(currentLocation, 16);
    }

    //Set GPS pin location
    MapLayer.SetPosition(gpsPin, currentLocation);

    //Update locations in view
    UpdateARView();
}
}
catch { }
});
}
catch { }
}

```

Visual Basic

```

Private Sub GpsPositionChanged(sender As Geolocator, args As PositionChangedEventArgs)
    If args.Position IsNot Nothing AndAlso args.Position.Coordinate IsNot Nothing
    AndAlso args.Position.Coordinate.Point IsNot Nothing Then
        GpsChanged(args.Position.Coordinate.Point.Position)
    End If
End Sub

Private Async Sub GpsChanged(position As BasicGeoposition)
    Try
        Await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
            Async Sub()
                Try
                    currentLocation = New Location(position.Latitude,
position.Longitude)
                    poiLocations = Nothing

                    'Update Location Data
                    If pinLayer IsNot Nothing Then
                        pinLayer.Children.Clear()

```

```

        Dim poi = Await NavteqPoiSearch(currentLocation)
        If poi IsNot Nothing AndAlso poi.ResultSet IsNot Nothing
Also poi.ResultSet.Results IsNot Nothing AndAlso poi.ResultSet.Results.Length > 0
Then
            poiLocations = poi.ResultSet.Results

            For Each r As Result In poiLocations
                Dim loc = New Location(r.Latitude, r.Longitude)

                Dim pin = New Pushpin()
                pin.Tag = r
                MapLayer.SetPosition(pin, loc)
                pinLayer.Children.Add(pin)
            Next

            'Center on users location
            MyMap.SetView(currentLocation, 16)
        End If

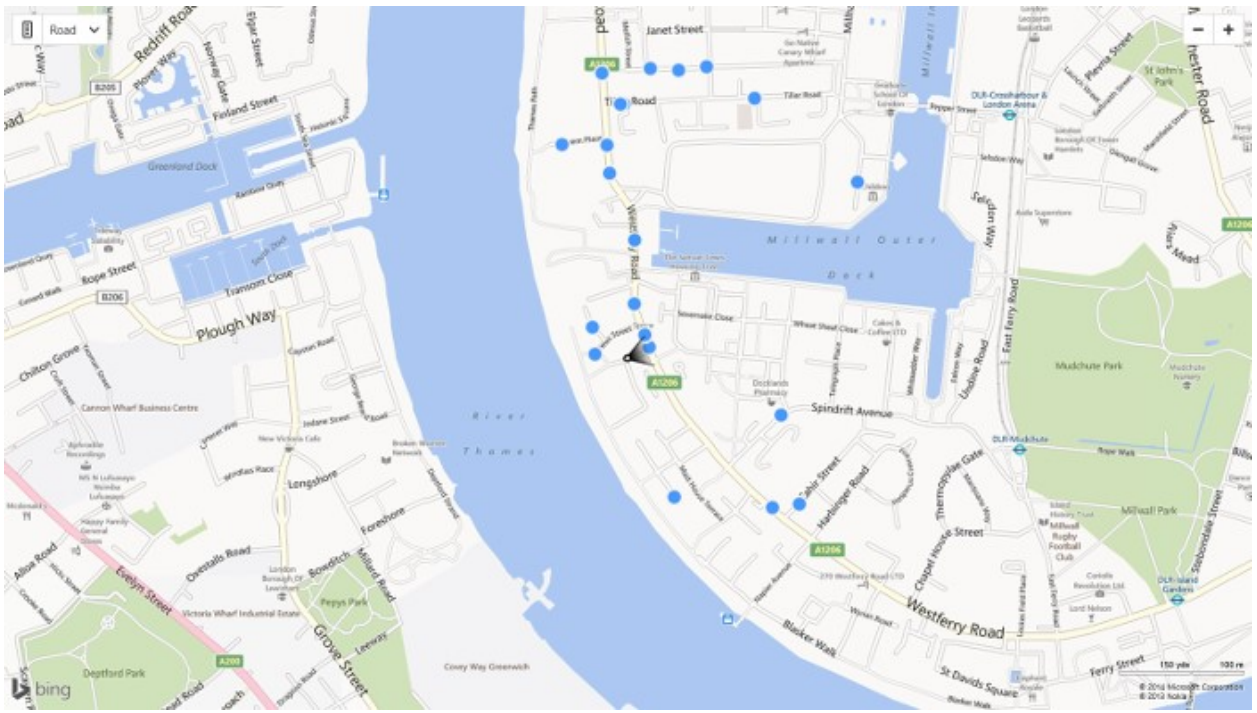
        'Set GPS pin location
        MapLayer.SetPosition(gpsPin, currentLocation)

        'Update locations in view
        UpdateARView()
    End If
Catch
End Try
End Sub)
Catch
End Try
End Sub

```

Testing the App

At this point the app is complete and ready to be tested. To deploy the app press F5 or the Debug button. The first time you run the app you will be promoted to allow the app access to your location and webcam. The app will then look for a rear facing camera. If it is able to find one it will stream video from it to the background of the augmented reality view. If a rear facing camera is not found a black background will be used instead. If your device is oriented in a face up or down position the map view will be displayed and look something like this.



If the device is not in a face up or down position the augmented reality view will be displayed. As you rotate about you should see various points of interest appear on the screen as the device points in their general direction. This view will look something like this.



Real World Example: SkyMap

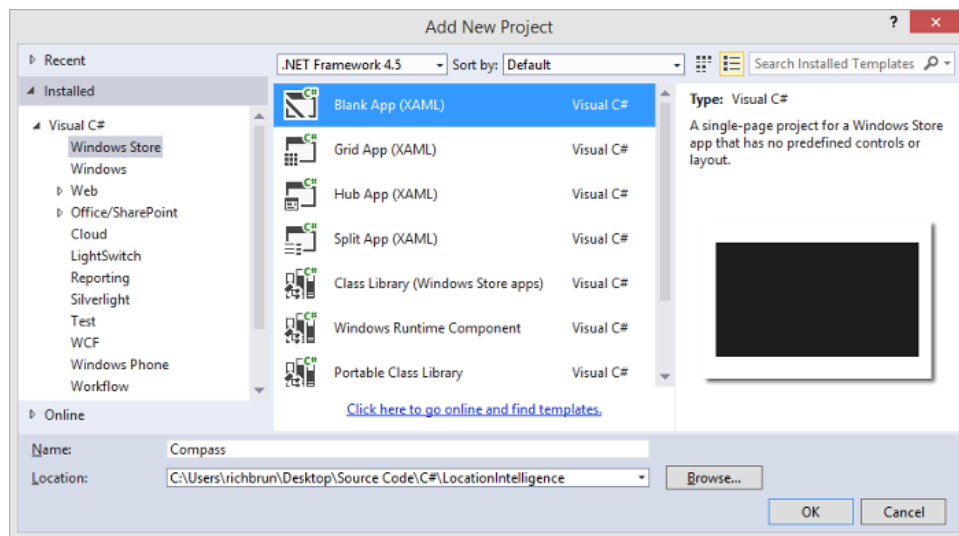
One of my favorite augmented reality app is called SkyMap. SkyMap turns your device into a virtual planetarium. Simply point your device at the sky and see what stars, constellations, planets or deep space objects are out there in real time. Tapping on any item on the screen provides you with additional details about that item. This is a fairly complex augmented reality app and makes use of the orientation sensor to accurately align the virtual objects on your devices screen with the real celestial objects in the sky. You can find this app in the Windows Store here: <http://bit.ly/1a26Xcr>. This app is also available for Windows Phone here: <http://bit.ly/1axdIpX>



Chapter 10: Creating a Templatable Compass Control

The compass sensor is easy to use and returns heading value between 0 and 360 in degrees relative to true or magnetic north. However not everyone understands headings in degrees and are likely more familiar with a handheld compass. In this chapter we are going to create a templatable compass control. This will allow us to easily change the look of the compass very easily in HTML or XAML. There are a number of different ways we can make the compass work as well. We could have a compass face that rotates such that readings are always aligned with their respective headings. Alternatively we could rotate a needle such that it points to the heading value on the compass that the device is facing.

To get started open up Visual Studios and create a new project in your preferred language; JavaScript, C# or Visual Basic. Select the **Blank App** template and call the application **Compass** and press **OK**.



Creating the Compass Control

The compass control will wrap the compass sensor and expose a bindable heading property. When the control loads we will attach an event handler to the reading change event of the compass sensor. When the control is unloaded we will detach this event to prevent any memory leaks. When the compass reading changes the heading property will be updated with the heading for magnetic north. The compass in some devices does not return a heading value for true north, magnetic north is the most likely to be available. In the JavaScript version of the compass control we will also expose a second property that will allow us to set the template of the control. If we want to rotate the compass face such that readings are always aligned with their respective headings we will have to rotate the compass face by the negative of the heading. Rather than exposing another property that exposes the negative value we can create a converter that modifies the heading value as needed through our binding from the compass template.

If you are using JavaScript right click on the **js** folder and select **Add -> New Item** and create a new JavaScript file called **compass-control.js**. Inside this file we will create a compass control using WinJS class framework (<http://bit.ly/1moPHhh>). Add the following code to the **compass-control.js** file.

```

(function () {
    "use strict";

    var compassControl = WinJS.Class.define(
        function(element, options) {
            this.element = element || document.createElement("div");
            this.element.winControl = this;

            // Set option defaults
            this._heading = 0;
            this._template = null;

            var self = this;

            var compassReadingChanged = function (e) {
                self.heading = e.reading.headingMagneticNorth;
            };

            var compass = Windows.Devices.Sensors.Compass.getDefault();
            if (compass) {
                compass.addEventListener("readingchanged", compassReadingChanged);
            }

            WinJS.Application.onunload += function () {
                if (compass) {
                    compass.removeEventListener("readingchanged",
compassReadingChanged);
                }
            };

            //Set user-defined options
            WinJS.UI.setOptions(this, options);
        },
        {
            heading: {
                get: function () {
                    return this._heading;
                },
                set: function (value) {
                    var oldValue = this._heading;
                    this._heading = value;
                    this.notify("heading", value, oldValue);
                }
            },
            template: {
                get: function () {
                    return this._template;
                },
                set: function (newTemplate) {
                    this._template = newTemplate;
                    this._template.winControl.render(this, this.element);
                }
            }
        }
    );

    //Make the compass control bindable control.
    WinJS.Class.mix(compassControl, WinJS.Binding.observableMixin);

    //Add the compass control and some compass related converters to a common Namespace.
    WinJS.Namespace.define("Compass", {
        CompassControl: compassControl,
    });

```

```

Converters: {
    rotationConverter: WinJS.Binding.converter(function (angle) {
        return 'rotate(' + angle + 'deg)';
    }),
    reverseRotationConverter: WinJS.Binding.converter(function (heading) {
        return 'rotate(' + (360 - heading) + 'deg)';
    })
}
});
})();

```

If you are using C# or Visual Basic right click on the project and select **Add -> New Item** and create a new class file called **ComapsssControl.cs**. The **CompassControl** class will derive from the **Control** class. In this class we will create a **DependencyProperty** for the heading so that we can easily bind to it. Update **CompassControl.cs** with the following code.

C#

```

using System;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

namespace Compass
{
    public class CompassControl : Control
    {
        private Windows.Devices.Sensors.Compass compass;

        public CompassControl()
        {
            this.DefaultStyleKey = typeof(CompassControl);

            this.Loaded += CompassControl_Loaded;
            this.Unloaded += CompassControl_Unloaded;
        }

        public static readonly DependencyProperty HeadingProperty =
            DependencyProperty.Register("Heading",
                typeof(double), typeof(CompassControl), new PropertyMetadata(0.0));

        public double Heading
        {
            get
            {
                return (double)GetValue(HeadingProperty);
            }
            set
            {
                SetValue(HeadingProperty, value);
            }
        }

        private void CompassControl_Loaded(object sender, RoutedEventArgs e)
        {
            compass = Windows.Devices.Sensors.Compass.GetDefault();
            if (compass != null)
            {
                compass.ReadingChanged += CompassReadingChanged;
            }
        }
    }
}

```

```

private void CompassControl_Unloaded(object sender, RoutedEventArgs e)
{
    if (compass != null)
    {
        compass.ReadingChanged -= CompassReadingChanged;
    }
}

private async void CompassReadingChanged(Windows.Devices.Sensors.Compass sender,
Windows.Devices.Sensors.CompassReadingChangedEventArgs args)
{
    try
    {
        await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
new Windows.UI.Core.DispatchedHandler(() =>
        {
            Heading = args.Reading.HeadingMagneticNorth;
        }));
    }
    catch { }
}
}
}

```

Visual Basic

```

Imports Windows.UI.Xaml
Imports Windows.UI.Xaml.Controls

Public Class CompassControl
    Inherits Control

    Private compass As Windows.Devices.Sensors.Compass

    Public Sub New()
        Me.DefaultStyleKey = GetType(CompassControl)

        AddHandler Me.Loaded, AddressOf CompassControl_Loaded
        AddHandler Me.Unloaded, AddressOf CompassControl_Unloaded
    End Sub

    Public Shared ReadOnly HeadingProperty As DependencyProperty =
DependencyProperty.Register("Heading", GetType(Double), GetType(CompassControl), New
PropertyMetadata(0.0))

    Public Property Heading() As Double
        Get
            Return Cdbl(GetValue(HeadingProperty))
        End Get
        Set(value As Double)
            SetValue(HeadingProperty, value)
        End Set
    End Property

    Private Sub CompassControl_Loaded(sender As Object, e As RoutedEventArgs)
        compass = Windows.Devices.Sensors.Compass.GetDefault()
        If compass IsNot Nothing Then
            AddHandler compass.ReadingChanged, AddressOf CompassReadingChanged
        End If
    End Sub

```

```

End Sub

Private Sub CompassControl_Unloaded(sender As Object, e As RoutedEventArgs)
    If compass IsNot Nothing Then
        RemoveHandler compass.ReadingChanged, AddressOf CompassReadingChanged
    End If
End Sub

Private Async Sub CompassReadingChanged(sender As Windows.Devices.Sensors.Compass,
    args As Windows.Devices.Sensors.CompassReadingChangedEventArgs)
    Try
        Await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, New
            Windows.UI.Core.DispatchedHandler(
                Sub()
                    Heading = args.Reading.HeadingMagneticNorth
                End Sub))
    Catch
    End Try
End Sub
End Class

```

If using C# or Visual Basic right click on the project and select **Add -> New Item** and create a new Class file called **ReverseRotationConverter.cs** and update it with the following code.

C#

```

using System;
using Windows.UI.Xaml.Data;

namespace Compass
{
    public class ReverseRotationConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, string
language)
        {
            if (value is double)
            {
                return 360 - (double)value;
            }

            return value;
        }

        public object ConvertBack(object value, Type targetType, object parameter,
string language)
        {
            if (value is double)
            {
                return 360 - (double)value;
            }

            return value;
        }
    }
}

```

Visual Basic

```
Imports Windows.UI.Xaml.Data

Public Class ReverseRotationConverter
    Implements IValueConverter

    Public Function Convert(value As Object, targetType As Type, parameter As Object,
        language As String) As Object Implements IValueConverter.Convert
        If TypeOf value Is Double Then
            Return 360 - Cdbl(value)
        End If

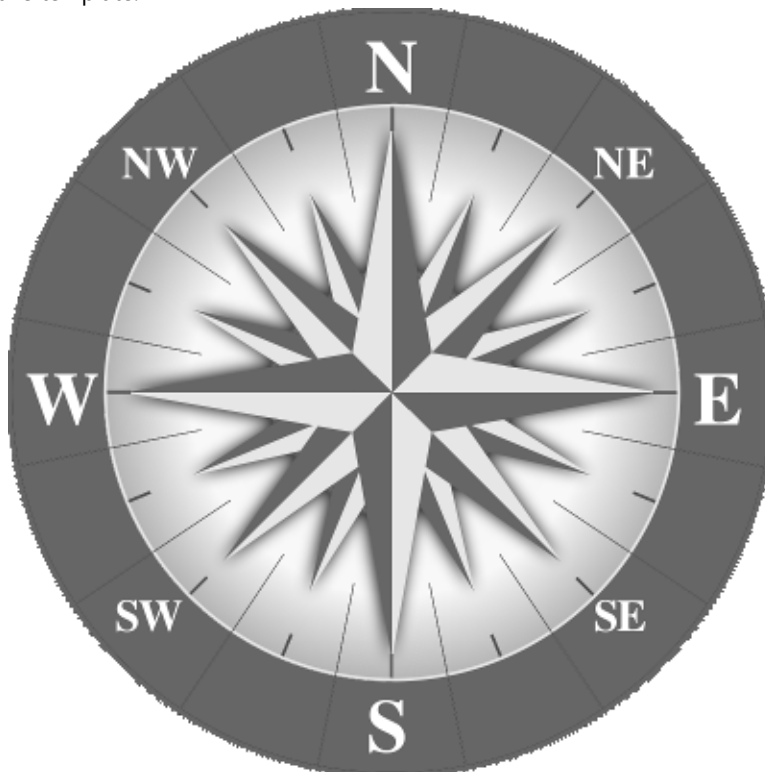
        Return value
    End Function

    Public Function ConvertBack(value As Object, targetType As Type, parameter As
        Object, language As String) As Object Implements IValueConverter.ConvertBack
        If TypeOf value Is Double Then
            Return 360 - Cdbl(value)
        End If

        Return value
    End Function
End Class
```

Creating the Compass Templates

We will create two different templates. For the first template we will create a compass where the face rotates such that all the compass headings stay pointing in the correct direction. We could create the compass face using pure XAML or a HTML5 Canvas but it's much faster and easier to use an image instead. We will use the following image as our compass face for this template.



CompassFace.png

Copy and paste these images from the code samples provided with this book into the **images** or **Assets** folder of your project. If using JavaScript open the **default.html** file and add the following HTML to the body of the page.

HTML

```
<div id="CompassTemplate" data-win-control="WinJS.Binding.Template">
  
</div>
```

If you are using C# or Visual Basic open the **App.xaml** file and update it with the following XAML.

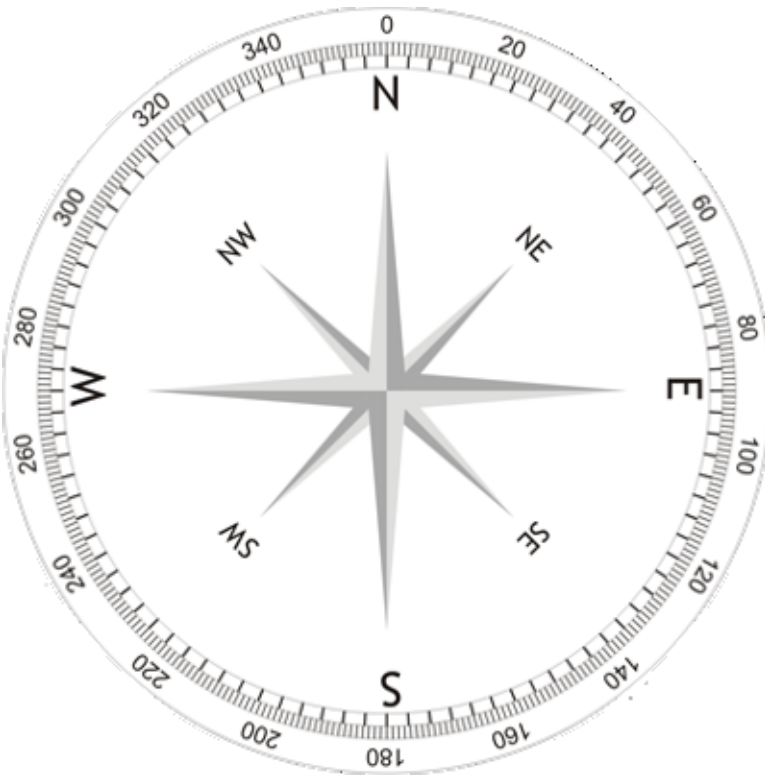
XAML

```
<Application
  x:Class="Compass.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:Compass">

  <Application.Resources>
    <local:ReverseRotationConverter x:Key="reverseRotationConverter"/>

    <Style TargetType="local:CompassControl">
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="local:CompassControl">
            <Image Source="ms-appx:///Assets/CompassFace.png" Stretch="None"
RenderTransformOrigin="0.5,0.5">
              <Image.RenderTransform>
                <RotateTransform Angle="{Binding Path=Heading,
RelativeSource={RelativeSource TemplatedParent}, Converter={StaticResource
reverseRotationConverter}}"/>
              </Image.RenderTransform>
            </Image>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </Application.Resources>
</Application>
```

The second compass template we will create consists of a needle overlaid on a compass face. The needle will rotate such that it points to the current heading on the compass face. For this template we will use the following two images.



CompassFace2.png



Needle.png

Copy and paste these images from the code samples provided with this book into the **images** or **Assets** folder of your project. If using JavaScript open the **default.html** file and add the following HTML to the body of the page.

HTML

```
<div id="NeedleCompassTemplate" data-win-control="WinJS.Binding.Template">
  <div class="needleCompass">
    
    
  </div>
</div>
```

If you are using C# or Visual Basic open the **App.xaml** file and add the following XAML to the **Application.Resources** section of the document.

XAML

```
<Style x:Name="NeedleCompassTemplate" TargetType="local:CompassControl">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="local:CompassControl">
        <Grid>
          <Image Source="ms-appx:///Assets/CompassFace2.png" Stretch="None"/>
          <Image Source="ms-appx:///Assets/Needle.png" Stretch="None"
RenderTransformOrigin="0.5,0.5"
          HorizontalAlignment="Center" VerticalAlignment="Center">
            <Image.RenderTransform>
```

```

        <RotateTransform Angle="{Binding Path=Heading,
RelativeSource={RelativeSource TemplatedParent}}"/>
    </Image.RenderTransform>
</Image>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>

```

Applying the Templates to the Compass Control

At this point we have all the components we need to add a nicely templatable compass control to our application. To implement the compass in JavaScript open the **default.css** file and update it with the following CSS styles.

```

.needleCompass {
    position:relative;
    width:400px;
    height:400px;
}

.needleCompass .compassNeedle {
    position:absolute;
    top:45px;
    left:185px;
}

.container {
    position:absolute;
    top:50%;
    left:50%;
    margin-top:-200px;
    margin-left:-425px;
}

```

Next open the **default.html** file. In here we will need to add a script reference to the compass-control.js file and add our two compasses to the page. Update this file with the following HTML.

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Compass</title>

    <!-- WinJS references -->
    <link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
    <script src="//Microsoft.WinJS.2.0/js/base.js"></script>
    <script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

    <!-- Compass references -->
    <link href="/css/default.css" rel="stylesheet" />
    <script src="/js/default.js"></script>

    <!-- Add reference to compas control -->
    <script src="/js/compass-control.js"></script>
</head>
<body>
    <!-- Compass Templates -->

```

```

<div id="CompassTemplate" data-win-control="WinJS.Binding.Template">
    
</div>

<div id="NeedleCompassTemplate" data-win-control="WinJS.Binding.Template">
    <div class="needleCompass">
        
        
    </div>
</div>

<div class="container">
    <!-- Add compass with rotating face to page -->
    <div data-win-control="Compass.CompassControl" data-win-
options="{template:select('#CompassTemplate')}" style="float:left;"></div>

    <!-- Add compass with needle to page -->
    <div data-win-control="Compass.CompassControl" data-win-
options="{template:select('#NeedleCompassTemplate')}" style="margin:0 0 0
50px;float:left;"></div>
</div>
</body>
</html>

```

If you are using C# or Visual Basic open the **MainPage.xaml** file and update it with the following XAML.

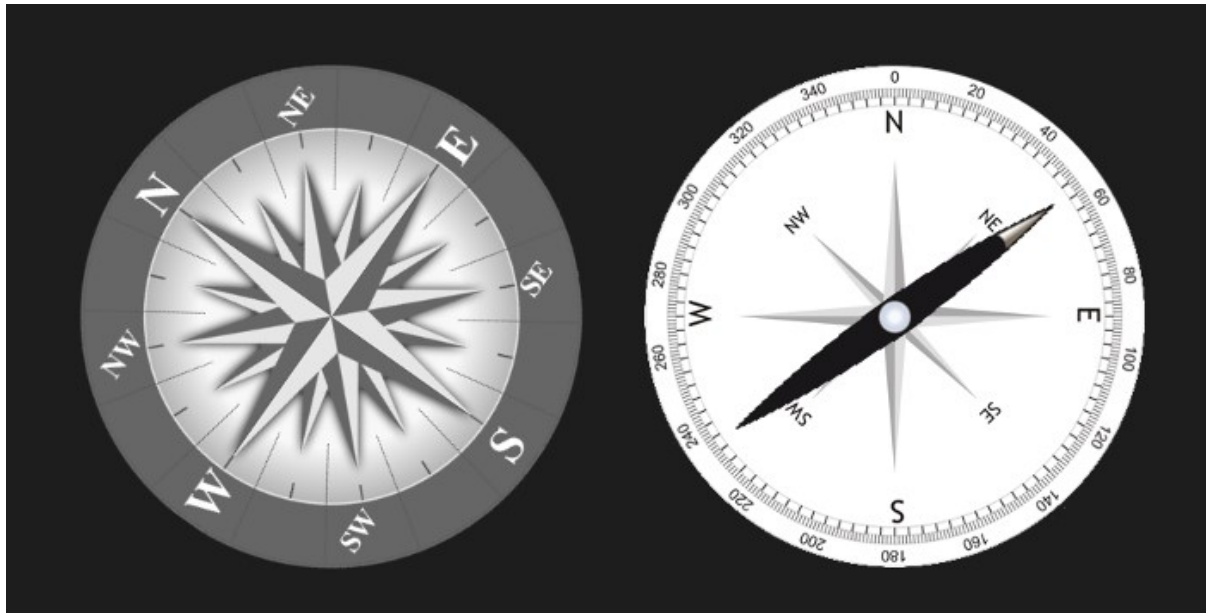
```

<Page
    x:Class="Compass.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:Compass"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <local:CompassControl/>
            <local:CompassControl Style="{StaticResource NeedleCompassTemplate}"
Margin="50,0,0,0"/>
        </StackPanel>
    </Grid>
</Page>

```

At this point the app is complete and ready to be tested. To deploy the app press F5 or the Debug button. When loaded two compasses will be displayed in the middle of the screen. As you rotate your device the compass face on the left will rotate such that its headings are always pointing in the correct direction. The needle of the compass on the right will rotate such that it points to the current heading the device is facing.



Chapter Summary

In this chapter you learned how to create templatable controls in HTML and XAML. The HTML control that was created used the WinJS class framework (<http://bit.ly/1moPHhh>).

As good as a compass is on its own, it is often useful to use it with a map. Try overlaying a compass over the map. Having it in the corner is handy, or give it some transparency and center it over the map. Since the compass control inherits from the **Control** class, and the **Control** class inherits from the **UIElement** class, this means you should be able to use the compass control as a pushpin in Bing Maps. This would be a good way to create a pushpin that shows the user's location and the direction they are facing.

Chapter 11: Cross Platform Development

The number of different platforms and devices that developers are creating apps for is constantly growing. Unfortunately so is the number of programming languages being used by these devices as well. You can create Windows Store and Windows Phone apps using .NET or JavaScript, iOS apps (iPhone and iPad) using Objective-C, Android apps using a version of Java, and Blackberry using a different version of Java. Many developers would love to be able to deploy their apps to all the major platforms but this rarely happens because of the amount of work involved in rewriting the apps for each platform.

Cross platform development allows you to write your app once and deploy it to several platforms. There are a number of different approaches to do this. In the first section of this chapter we will look at using JavaScript to create cross platform applications and some of the tools available to make this easier. In the second half of this chapter we will take a look at using .NET to create a cross platform app that targets Windows Store and Windows Phone. In both sections we are going to build a simple map application that allows the user to display their location and search for other locations.

Creating a Cross Platform app using JavaScript

JavaScript has been around for close to 20 years now and has primarily been used in web pages. In recent years we have seen JavaScript used in a number of other places such as server side code, and in apps. As we have seen throughout this book you can create Windows Store apps using JavaScript. Did you know that you can also create apps for other platforms using JavaScript? Most platforms don't support native apps written in JavaScript, but most platforms do have a web browser control that can be embedded into a native app. This means that you can create a mobile app in JavaScript and host it inside of the web browser control of a native app. This approach for creating cross platform apps has really started taking off in the last few years.

As we already seen we can use the Bing Maps Windows Store SDK to add maps to a Windows Store app using JavaScript. One nice feature of the Bing Maps Windows Store SDK is that it is nearly identical to the Bing Maps V7 AJAX map control which is a web based JavaScript map control. In addition to this the Bing Maps V7 AJAX control is supported on all major PC and Mac browsers, Windows Phone 8, iOS, Android, and Blackberry 6 and above. Also, since Kindle Fire is essentially a heavily modified version of Android, it works on their too.

When creating cross platform apps using JavaScript you will often find that not all devices will have the same capabilities. Some will have support for one or two touch points well others will have none. Rather than writing code that target specific browsers or operating systems to write code that handles each case it is better to instead test for features in JavaScript. This will increase the chances of your code continuing to work in future versions of web browsers. Since it's likely that not all users will have a device that supports multi-touch it would be a good idea to provide zooming or Birdseye rotation buttons to the app as the user won't be able to pinch their screen to zoom the map or rotate it with two fingers to view the Birdseye maps from a different angle. Here is an example of how to test for multi-touch touch support in JavaScript.

```
if ('ongesturechange' in window || //iOS & android
    (navigator.msMaxTouchPoints && navigator.msMaxTouchPoints > 1)) //Win8, WP8 & IE
{
    //Device supports multi-touch
}
```

If the device doesn't support single touch either than we would also want to include panning buttons as well otherwise the user won't be able to move the map. Here is an example of how to test for single touch support in JavaScript.

```
if ('ontouchstart' in window || //iOS & android
    (navigator.msMaxTouchPoints && navigator.msMaxTouchPoints > 0)) //Win8, WP8 & IE
{
    //Device supports single touch
}
```

Sometimes features may appear available but there may be a better option to use. For example, the JavaScript alert method is useful for sending notifications to the user in the app, however in Windows Store apps the **alert** function is undefined, instead we have to use the **MessageDialog** function. A simple way to deal with this in a cross platform app is to create a reusable function that checks if the **MessageDialog** function is available. If it is then use it otherwise fall back to the **alert** function. Here is an example of how you can do this.

```
function showMessage(msg) {
    if (typeof Windows != "undefined" &&
        Windows.UI != null &&
        Windows.UI.Popups != null) {
        var popup = Windows.UI.Popups.MessageDialog(msg);
        popup.showAsync();
    } else {
        alert(msg);
    }
}
```

Another other common task in JavaScript is to add events to DOM elements. Not all browsers handle this the same. Most use the **addEventListener** function, however some older browser use the **attachEvent** function. We can easily create a simple function that tests to see if the **addEventListener** function exists and if it doesn't then fall back onto the **attachEvent** function. Here is an example of how you can do this.

```
function addListener(element, eventName, eventHandler) {
    if (element.addEventListener) {
        element.addEventListener(eventName, eventHandler, false);
    } else if (element.attachEvent) {
        if (eventName == 'DOMContentLoaded') {
            eventName = 'readystatechange';
        }
        element.attachEvent('on' + eventName, eventHandler);
    }
}
```

When targeting mobile devices you will find that not all screens have the same pixel density. In fact most mobile devices have higher pixel densities than most computers. As such we need to scale the viewport of the app such that it is equal to the device width. This can be done in Internet Explorer by using the **@-ms-viewport** CSS style while other browsers prefer to use a metadata tag in the header of the page to do this.

```
<meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1, minimum-scale=1, width=device-width, height=device-height, target-densitydpi=device-dpi" />
```

Unfortunately there isn't an easy way to distinguish between the desktop and mobile version of Internet Explorer 10 and 11 in CSS. When using the desktop version of Internet Explorer we need to set the viewport using a style of **@-ms-viewport{width:auto!important}**. This can be done by adding the following JavaScript to the head of the page.

```
<script type="text/javascript">
    //Work around for scaling bug between IE and IE mobile Desktop
```

```

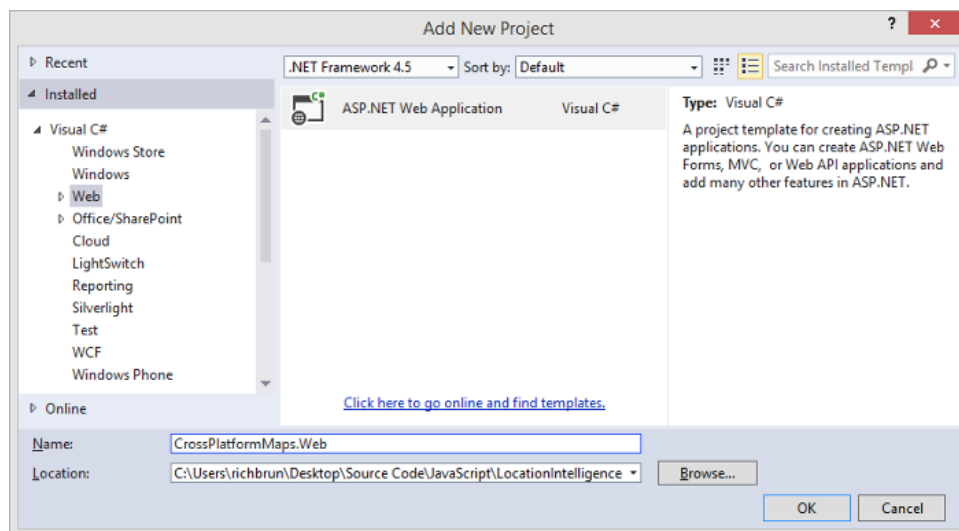
(function () {
    var n = navigator;
    var ie = new RegExp("Trident");
    var ieMobile = new RegExp("IEMobile");
    if (ie.test(navigator.userAgent) &&
        !ieMobile.test(navigator.userAgent)) {
        var msViewportStyle = document.createElement("style");
        try{
            msViewportStyle.appendChild(
                document.createTextNode("@-ms-viewport{width:auto!important}")
            );
        }catch(e){
            msViewportStyle.text = '@-ms-viewport{width:auto!important}';
        }
        document.getElementsByTagName("head")[0].appendChild(msViewportStyle);
    }
})();
</script>

```

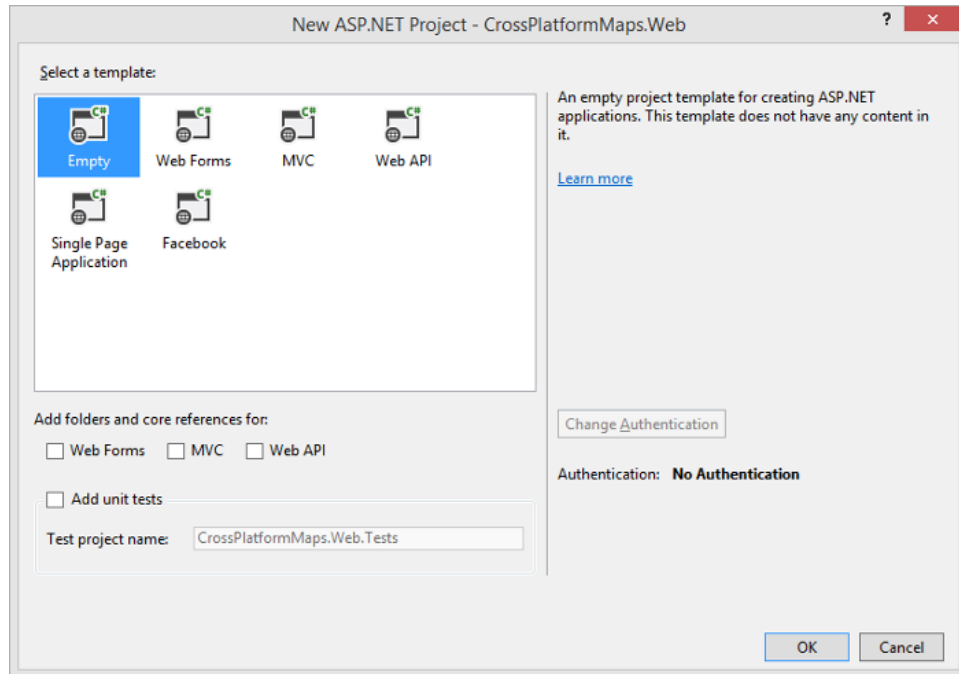
Creating a Mobile Web App

To get started we are going to create a mobile friendly web app which will work right in the browser. There are a number of benefits to starting here. First, we can ensure we are using pure JavaScript and don't accidentally rely too heavily on the WinJS libraries that are in Windows Stores apps. It is also easy to test in a browser and host online so you can test on mobile browsers as well.

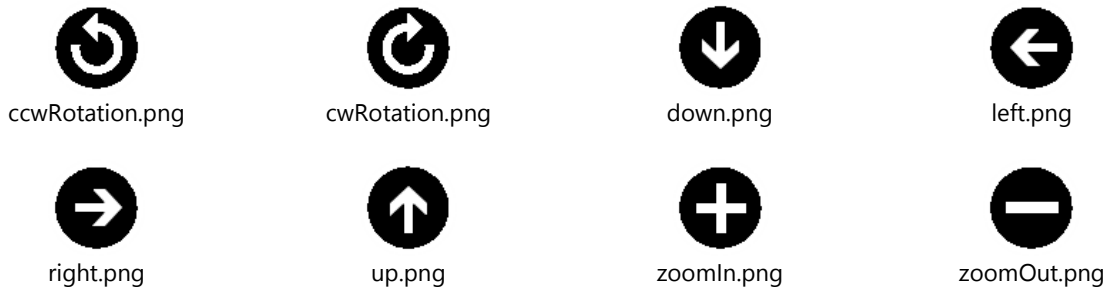
Open up Visual Studios and create a new project that uses the ASP.NET Web Application template that is under the Visual C# > Web section. Call the application **CrossPlatformMaps.Web** and press **OK**.



A window will open where you can select the ASP.NET template to create. Select the **Empty** template and press OK.



In the project create three folders called **css**, **images** and **js** by right clicking on the project and selecting **Add -> New Folder**. Next add an HTML file to the project called **index.html**. Create a new Style Sheet called **map.css** and add it to the **css** folder. Create a JavaScript file called **map.js** and add it to the **js** folder. As mentioned earlier in this chapter there may be times where we want to provide the user with pan, zoom and Birdseye rotation buttons. Bing Maps already has a navigation bar with all these buttons but on high pixel density screens, which are often used on mobile devices, the navigation bar tends to be too small and hard to use with touch, as such the following images will be used for these buttons and should be added to the **images** folder of the project.



We will also need some buttons for when the user wants to see their location on the map, do a search or change the map style. These buttons will be white and be displayed above a black bar, similar to what is done in Windows Phone apps. Add the following images to the **images** folder.



Now open the **index.html** file. In here we will need to add references to all the CSS and JavaScript files we created earlier along with a script reference to the Bing Maps V7 AJAX control. We will also add a **div**'s for the map, a search

panel and map mode panel. We will also overlay the buttons over top the map. To do this update the **index.html** file with the following HTML.

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1,
minimum-scale=1, width=device-width, height=device-height, target-densitydpi=device-dpi" />

    <link rel="stylesheet" type="text/css" href="css/map.css" />

    <script type="text/javascript">
      //Work around for scaling bug between IE and IE mobile Desktop
      (function () {
        var n = navigator;
        var ie = new RegExp("Trident");
        var ieMobile = new RegExp("IEMobile");
        if (ie.test(navigator.userAgent) &&
!ieMobile.test(navigator.userAgent)) {
          var msViewportStyle = document.createElement("style");
          try{
            msViewportStyle.appendChild(
              document.createTextNode("@-ms-viewport{width:auto!important}")
            );
          }catch(e){
            msViewportStyle.text = '@-ms-viewport{width:auto!important}';
          }
          document.getElementsByTagName("head")[0].appendChild(msViewportStyle);
        }
      })();
    </script>

    <script type="text/javascript"
src="http://ecn.dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=7.0"></script>

    <script type="text/javascript" src="js/map.js"></script>
  </head>
  <body>
    <div id="mapContainer">
      <div id='myMap'></div>
      <div class="appBar">
        <div class="appBarContainer">
          
          
          
        </div>
      </div>
      <div id="navBar">
        
        
        
        
      </div>
      <div id="zoomRotateBar">
        
        
        <div id="rotationBtns">
          
          
        </div>
      </div>
    </div>
  </body>
</html>
```

```

        </div>
    </div>
</div>

<div id="searchPanel" style="display:none;">
    <div>
        <h2>Search</h2>
        <input type="text" id="searchTbx" /></td>
        
    </div>
</div>

<div id="mapModePanel" style="display:none;">
    <div>
        <h2>Map Mode</h2>

        <ul>
            <li>
                <input type="radio" name="mapMode" id="autoMode" value="auto"
checked="checked" />
                <label for="autoMode">Auto</label>
            </li>
            <li>
                <input type="radio" name="mapMode" id="aerialMode" value="aerial" />
                <label for="aerialMode">Aerial</label>
            </li>
            <li>
                <input type="radio" name="mapMode" id="birdseyeMode"
value="birdseye" />
                <label for="birdseyeMode">Birdseye</label>
            </li>
            <li>
                <input type="radio" name="mapMode" id="roadMode" value="road" />
                <label for="roadMode">Road</label>
            </li>
        </ul>
    </div>
</div>
</body>
</html>

```

The HTML in the **index.html** file defines the structure of the page but we will need to add some CSS styles to the **map.css** file so that this page is laid out properly. These styles will also need to be able to define the layout of the app when the orientation of the device changes. Open the **map.css** file and update it with the following CSS styles.

```

/* Global Styles */

html, body {
    height: 100%;
    width: 100%;
    margin: 0px;
    padding: 0px;
    overflow: hidden;
    font-family: "Segoe UI", "Segoe WP", Helvetica, sans-serif;
    font-weight: normal;
}

h2 {
    font-size: 22pt;
    margin: 10px;
}

```

```

}

body {
  font-size: 9pt;
  letter-spacing: 0.02em;
  background-color: #000000;
  color: #ffffff;
  margin-left: 24px;
}

input[type="radio"] {
  width:20px;
  height:20px;
  margin:5px 0px;
}

input[type="text"]
{
  font-size: 20px;
  line-height: 24px;
  padding:5px;
  margin:5px;
}

/* Custom Styles */

#myMap {
  position:absolute;
  top:0;
  left:0;
  width:100%;
  height:100%;
}

#zoomRotateBar {
  position:absolute;
  top:10px;
  left:10px;
}

#navBar {
  position:absolute;
  top:60px;
  left:10px;
  width:55px;
}

.appBar {
  position:absolute;
  overflow:hidden;
  background-color:black;
  top:0;
  right:0;
  width:55px;
  height:100%;
}

.appBarContainer {
  position:absolute;
  top:50%;
}

```

```

#rotationBtns {
    float:left;
}

.appBar img, #navBar img, #zoomRotateBar img {
    width: 35px;
    height: 35px;
    margin: 10px 10px 5px 10px;
    float:left;
}

.appBar img:active, #navBar img:active, #zoomRotateBar img:active {
    background-color:#e8e8e8;
    background-clip: border-box;
    box-sizing: border-box;
    border-radius: 50%;
}

#mapModePanel, #searchPanel {
    position:absolute;
    top:0px;
    left:0px;
    width:100%;
    height:100%;
    background-color:#000;
    opacity:0.8;
    filter:alpha(opacity=80);
}

#mapModePanel div, #searchPanel div {
    position:relative;
    top:calc(50% - 80px);
    margin:auto;
    width:380px;
}

#mapModePanel ul {
    list-style-type:none;
    margin:0 0 0 10px;
    padding:0px;
}

#mapModePanel ul label {
    font-size:25px;
}

#mapModePanel li{
    list-style-type:none;
    margin:0px;
}

#searchPanel input {
    margin: 5px;
    float:left;
}

#searchPanel img {
    width: 35px;
    height: 35px;
}

```

```

    margin: 5px 0px 0px 10px;
    float:left;
}

/* Orientation based styles */
@media all and (orientation:portrait) {
    @-ms-viewport {
        width: 320px;
        user-zoom: fixed;
        max-zoom: 1;
        min-zoom: 1;
    }

    #myMap {
        bottom:55px;
        right:0;
    }

    .appBar {
        top:auto;
        right:auto;
        bottom:0;
        left:0;
        width:100%;
        height:55px;
    }

    .appBar .appBarContainer {
        position:relative;
        width:180px;
        top:0;
        margin: 0 auto;
    }
}

@media all and (orientation:landscape) {
    @-ms-viewport {
        width:480px;
        user-zoom: fixed;
        max-zoom: 1;
        min-zoom: 1;
    }

    #myMap {
        bottom:0;
        right:55px;
    }

    .appBar {
        top:0;
        right:0;
        width:55px;
        height:100%;
        display:table;
    }

    .appBar .appBarContainer {
        margin-top: -90px;
    }
}

```

```

@media screen and (-ms-view-state: fullscreen-landscape),
  screen and (-ms-view-state: fullscreen-landscape),
  screen and (-ms-view-state: filled) {
  @-ms-viewport {
    width:100%;
    user-zoom: fixed;
    max-zoom: 0;
    min-zoom: 0;
  }
}

```

The final step is to add all the JavaScript that powers the app. When the app loads it will first need to load the map then determine if single or multi-touch is supported by the devices and display additional navigation buttons as required. The app will then need to attach click events to all the buttons that are displayed. In addition to this we will also need JavaScript that uses the **GeoLocationProvider** in Bing Maps to get the user's location and the **Search** module to geocode users queries. Open the **map.js** file and update it with the following JavaScript.

```

(function () {
  var map,
    bingMapsKey = "YOUR_BING_MAPS_KEY",
    searchManager,
    geoLocationProvider,
    gpsLayer,
    gpsEnabled = false;

  function init() {
    //Check to see if the Windows 8 version of the map control is being used by checking
    for the ClientRegion property on the map control.
    if (Microsoft.Maps.ClientRegion) {
      Microsoft.Maps.loadModule('Microsoft.Maps.Map', { callback: getMap });
    } else {
      getMap();
    }

    //Test for multi-touch support
    if ('ongesturechange' in window || //iOS & android
      (navigator.msMaxTouchPoints && navigator.msMaxTouchPoints > 1)) //Win8, WP8 & IE
    {
      //Hide zoom and rotate controls
      document.getElementById('zoomRotateBar').style.display = 'none';
    } else {
      //Only display rotation buttons when the map style is birdseye
      Microsoft.Maps.Events.addHandler(map, 'maptypechanged', updateNavBar);
      Microsoft.Maps.Events.addHandler(map, 'viewchangeend', updateNavBar);

      //Add zooming and rotating functionality
      addListener(document.getElementById('zoomInBtn'), 'click', function () {
        map.setView({ zoom: map.getZoom() + 1 });
      });

      addListener(document.getElementById('zoomOutBtn'), 'click', function () {
        map.setView({ zoom: map.getZoom() - 1 });
      });

      addListener(document.getElementById('rotateCWBtn'), 'click', function () {
        map.setView({ heading: map.getHeading() + 90 });
      });

      addListener(document.getElementById('rotateCCWBtn'), 'click', function () {

```

```

        map.setView({ heading: map.getHeading() - 90 });
    });
}

//Test for single-touch support
if ('ontouchstart' in window || //iOS & android
    (navigator.msMaxTouchPoints && navigator.msMaxTouchPoints > 0)) //Win8, WP8 & IE
{
    //Hide navigation controls
    document.getElementById('navBar').style.display = 'none';
} else {
    //Add panning functionality
    addListener(document.getElementById('upBtn'), 'click', function () {
        map.setView({ center: map.getCenter(), centerOffset: new
Microsoft.Maps.Point(0, 100) });
    });

    addListener(document.getElementById('leftBtn'), 'click', function () {
        map.setView({ center: map.getCenter(), centerOffset: new
Microsoft.Maps.Point(100, 0) });
    });

    addListener(document.getElementById('rightBtn'), 'click', function () {
        map.setView({ center: map.getCenter(), centerOffset: new
Microsoft.Maps.Point(-100, 0) });
    });

    addListener(document.getElementById('downBtn'), 'click', function () {
        map.setView({ center: map.getCenter(), centerOffset: new
Microsoft.Maps.Point(0, -100) });
    });
}

addListener(document.getElementById('searchBtn'), 'click', function () {
    document.getElementById('searchPanel').style.display = '';
    document.getElementById('searchTbx').focus();
});

addListener(document.getElementById('searchTbx'), 'keydown', function (e) {
    if (!e) {
        e = window.event;
    }

    //process search when enter key pressed
    if (e.keyCode == 13) {
        search(this.value);
    }
});

addListener(document.getElementById('geocodeBtn'), 'click', function () {
    search(document.getElementById('searchTbx').value);
});

addListener(document.getElementById('gpsBtn'), 'click', toggleGPS);

addListener(document.getElementById('mapModeBtn'), 'click', function () {
    document.getElementById('mapModePanel').style.display = '';
});

var mapModeBtns = document.getElementsByName('mapMode');
```

```

    for (var i = 0; i < mapModeBtns.length; i++) {
        addListener(mapModeBtns[i], 'click', function () {
            setMapMode(this.value);

            document.getElementById('mapModePanel').style.display = 'none';
        });
    }
}

function getMap() {
    var mapOptions = {
        credentials: bingMapsKey,
        showDashboard: false,
        showCopyright: false,
        showScalebar: false,
        enableSearchLogo: false,
        enableClickableLogo: false,
        backgroundColor: new Microsoft.Maps.Color(255, 0, 0, 0)
    };

    //Initialize the map
    map = new Microsoft.Maps.Map(document.getElementById("myMap"), mapOptions);

    gpsLayer = new Microsoft.Maps.EntityCollection();
    map.entities.push(gpsLayer);
}

function updateNavBar() {
    if (map.isRotationEnabled()) {
        document.getElementById('rotationBtns').style.display = '';
    } else {
        document.getElementById('rotationBtns').style.display = 'none';
    }
}

function search(query) {
    if (searchManager) {
        var request = {
            where: query,
            count: 1,
            callback: function (response) {
                if (response &&
                    response.results &&
                    response.results.length > 0) {
                    var r = response.results[0];
                    var l = new Microsoft.Maps.Location(r.location.latitude,
r.location.longitude);

                    //Display result on map
                    var p = new Microsoft.Maps.Pushpin(l);
                    map.entities.push(p);

                    //Zoom to result
                    map.setView({ center: l, zoom: 15 });
                } else {
                    showMessage("Not results found.");
                }

                document.getElementById('searchPanel').style.display = 'none';
            },
            errorCallback: function(request) {

```



```

        showMessage("Unable to Geocode request.");
        document.getElementById('searchPanel').style.display = 'none';
    }
};

searchManager.geocode(request);
} else {
    //Load the Search module and create a search manager.
    Microsoft.Maps.loadModule('Microsoft.Maps.Search', {
        callback: function () {
            //Create the search manager
            searchManager = new Microsoft.Maps.Search.SearchManager(map);

            //Perform search logic
            search(query);
        }
    });
}
}

function toggleGPS() {
    gpsEnabled = !gpsEnabled;

    // Initialize the location provider
    if (!geoLocationProvider) {
        geoLocationProvider = new Microsoft.Maps.GeoLocationProvider(map);
    }

    //Clear the GPS layer
    gpsLayer.clear();

    if (gpsEnabled) {
        // Get the user's current location
        geoLocationProvider.getCurrentPosition({
            successCallback: function (e) {
                gpsLayer.push(new Microsoft.Maps.Pushpin(e.center));
            },
            errorCallback: function (e) {
                showMessage(e.internalError);
            }
        });
    } else {
        //Remove the accuracy circle and cancel any request that might be processing
        geoLocationProvider.removeAccuracyCircle();
        geoLocationProvider.cancelCurrentRequest();
    }
}

function setMapMode(mode) {
    var m;

    switch (mode) {
        case 'auto':
            m = Microsoft.Maps.MapTypeId.auto;
            break;
        case 'aerial':
            m = Microsoft.Maps.MapTypeId.aerial;
            break;
        case 'birdseye':
            m = Microsoft.Maps.MapTypeId.birdseye;
            break;
    }
}

```

```

        case 'road':
        default:
            m = Microsoft.Maps.MapTypeId.road;
            break;
    }

    map.setView({ mapTypeId: m });
}

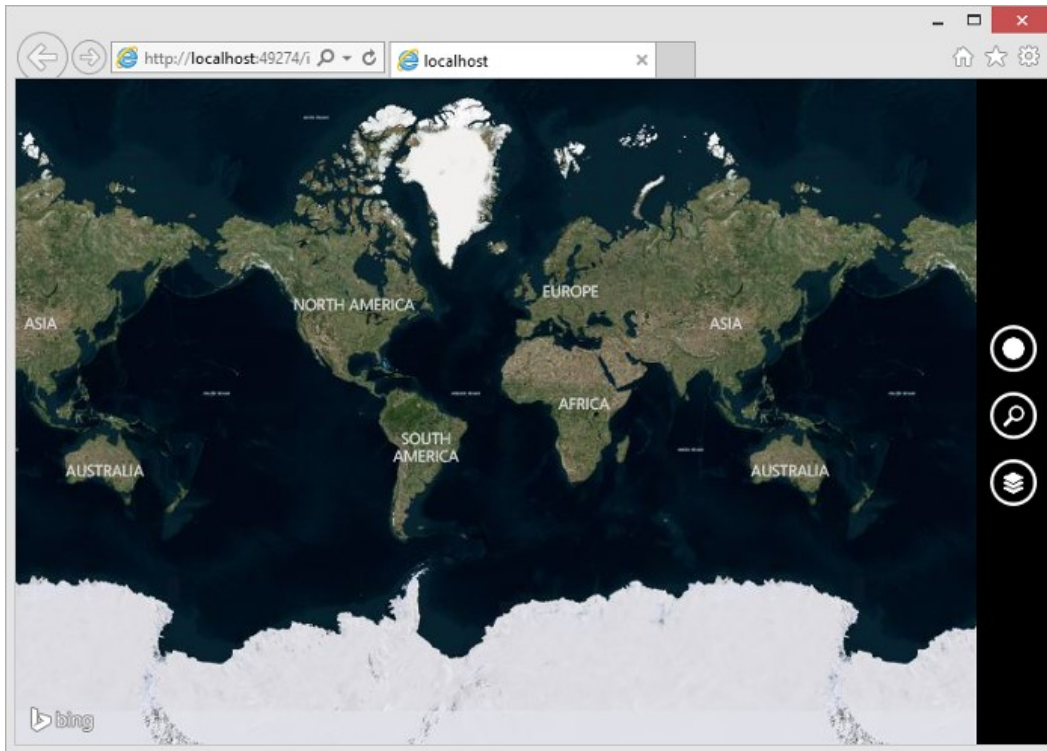
function showMessage(msg) {
    if (typeof Windows != "undefined" &&
        Windows.UI != null &&
        Windows.UI.Popups != null) {
        var popup = Windows.UI.Popups.MessageDialog(msg);
        popup.showAsync();
    } else {
        alert(msg);
    }
}

//Cross browser support for adding events. Mainly for IE7/8
function addListener(element, eventName, eventHandler) {
    if (element.addEventListener) {
        element.addEventListener(eventName, eventHandler, false);
    } else if (element.attachEvent) {
        if (eventName == 'DOMContentLoaded') {
            eventName = 'readystatechange';
        }
        element.attachEvent('on' + eventName, eventHandler);
    }
}

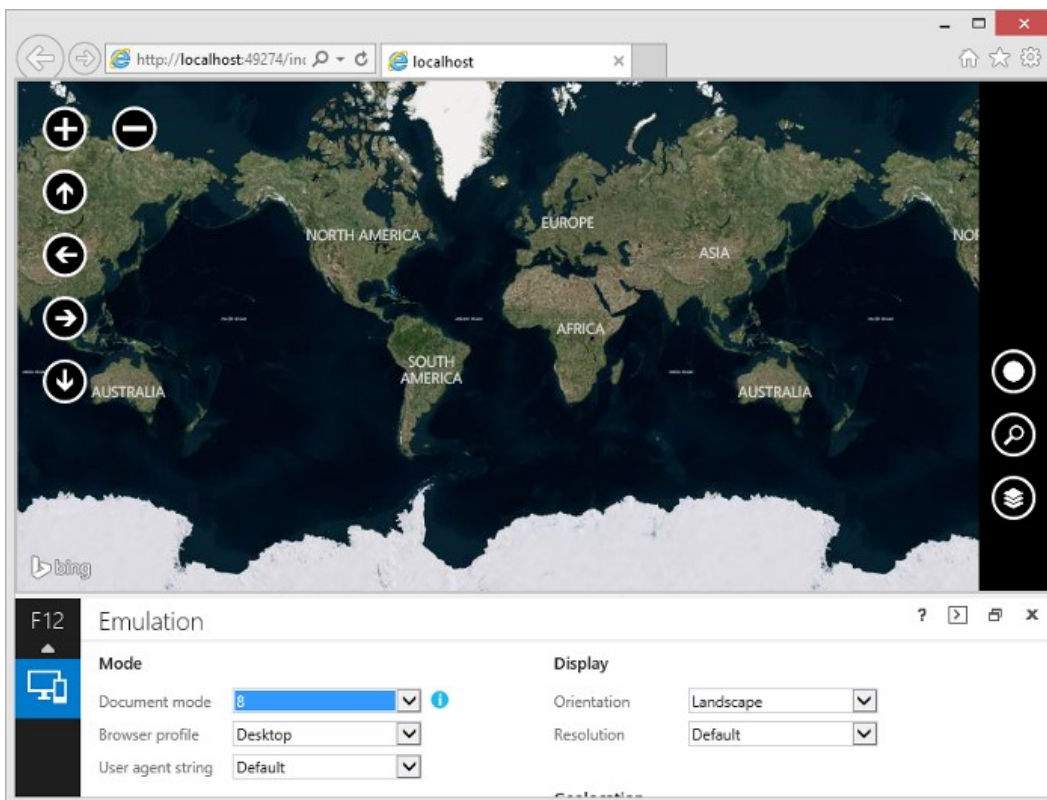
addListener(document, 'DOMContentLoaded', init);
})();

```

At this point the mobile web app is complete. Open the **index.html** file in a browser to try it out. This is what the app looks like when using a device and browser that support multi-touch.



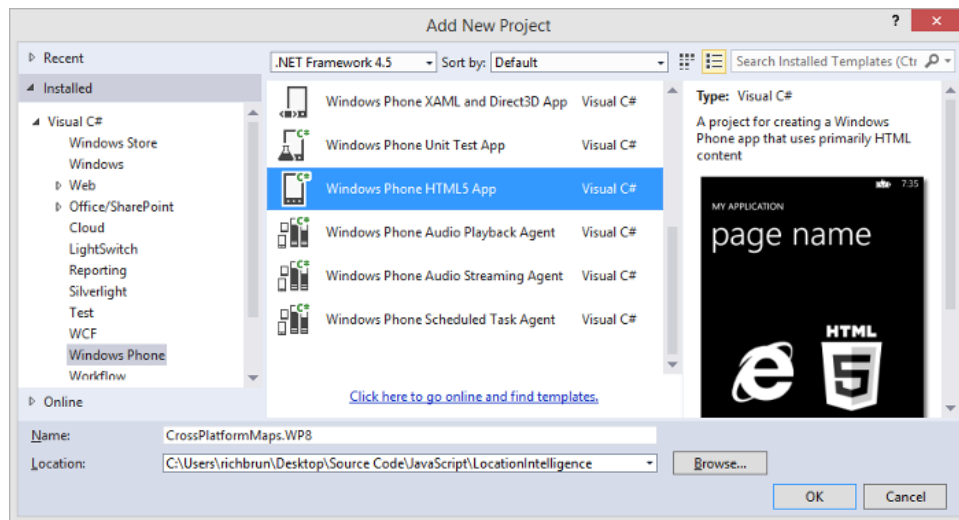
Here is the app again being used in a browser that doesn't support touch (IE8).



Tip: In Internet Explorer 11 you can easily emulate different browsers to see how your app works in them. Simply press F12 when Internet Explorer is open and it will display the developer's tools to you.

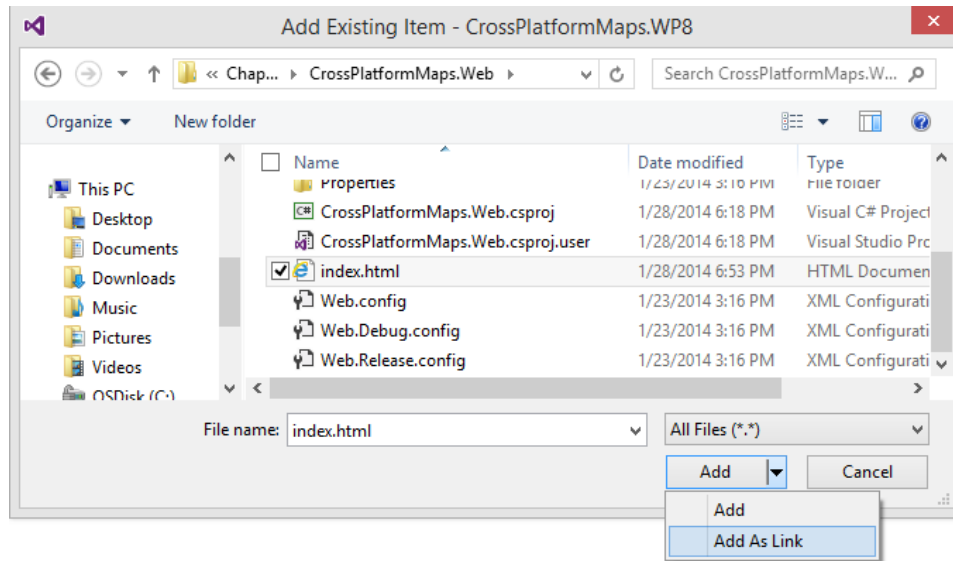
Integrating into a Windows Phone App

Integrating this mobile friendly web page into a Windows Phone App is fairly easy. First create a new project by right clicking on the solution and selecting **Add -> New Project**. Choose your preferred development language: C# or Visual Basic. From the Windows Phone templates section select the **Windows Phone HTML5 App** template and set the name to **CrossPlatformMaps.WP8**.

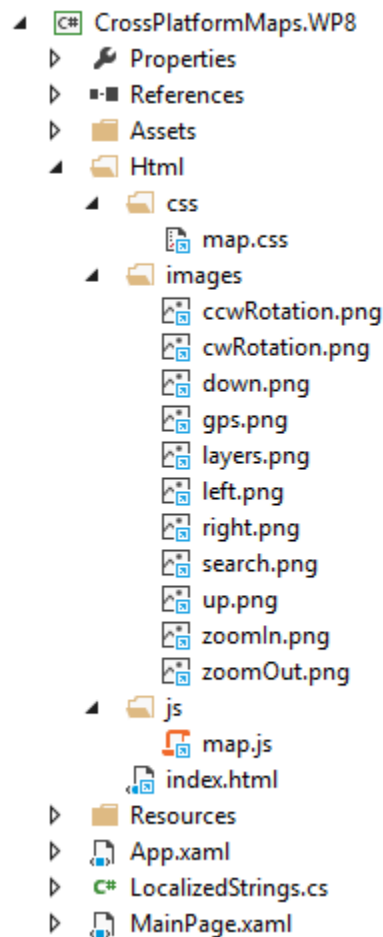


If you are concerned about having to use C# or Visual Basic when creating a JavaScript app, don't worry there are only a couple of lines of code in this language that we need. Majority of the app will be in JavaScript.

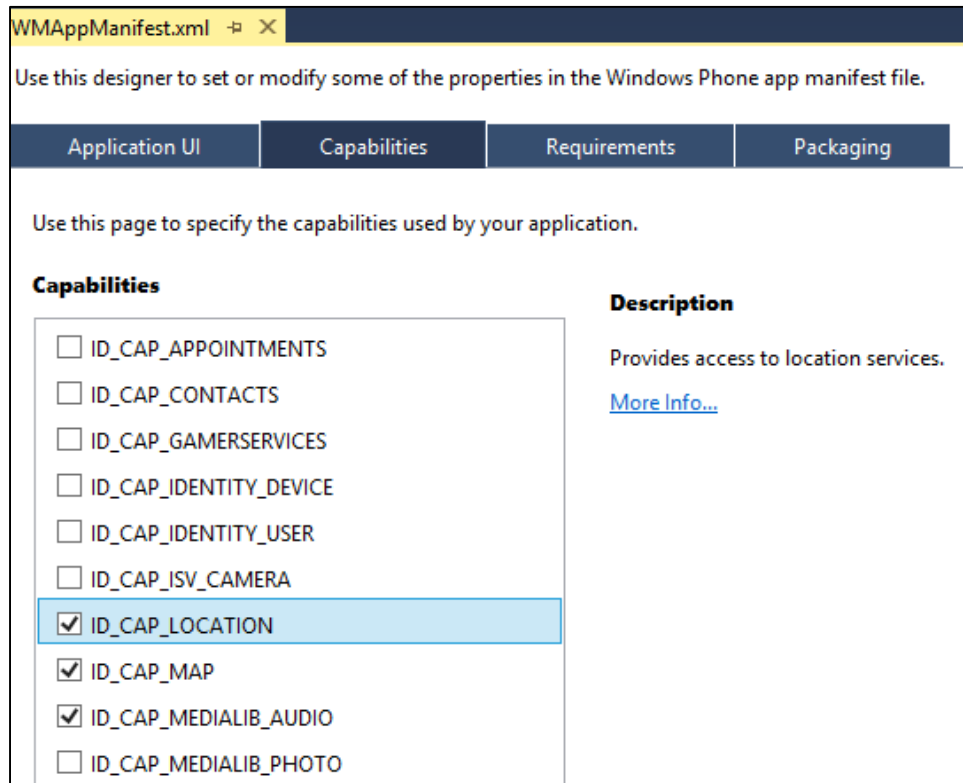
The generated project is pretty straight forward. In the **MainPage.xaml** there is a **WebBrowser** control that is stretched to fill the screen and an **AppBar**. Delete the XAML for the **AppBar**. We won't need this as we have already created an app bar in HTML. In the **MainPage.xaml.cs** the load event handler for when the **WebBrowser** is triggered it enables JavaScript in the browser and then navigates to a webpage that is embedded in the project. In the **Html** folder of the project is a simple webpage called **index.html**. Delete the **index.html** file. Add three folders to the **Html** folder called **css**, **images** and **js**. Do this by right clicking on the **Html** folder and selecting **Add -> New Folder**. Next click on the **Html** folder and select **Add -> Existing Item**. Navigate to the **CrossPlatformMaps.Web** project and select the **index.html** file. Instead of pressing the **Add** button, click on the arrow beside the **Add** button and select **Add As Link**. This will make the project reference the existing file from the other project. This makes things easier to manage as changing the file in either project will automatically update it in the other project.



Repeat this process to reference in all the CSS, images and JavaScript files from the web project into their respective folders in the WP8 project. At this point your project should look like this.



Since our app will need access to the user's location to show it on the map we have to set this capability in the app. To do this, double click on the **WMAppManifest.xml** file which is located in the **Properties** folder. Select the Capabilities tab and check the ID_CAP_Location option.



One final step is to give the **WebBrowser** control access to the user's location by setting the **IsGeolocationEnabled** property of the browser to true. To do this open the **MainPage.xaml.cs** file and update the **Browser_Loaded** event handler with the following code.

C#

```
private void Browser_Loaded(object sender, RoutedEventArgs e)
{
    Browser.IsScriptEnabled = true;
    Browser.IsGeolocationEnabled = true;

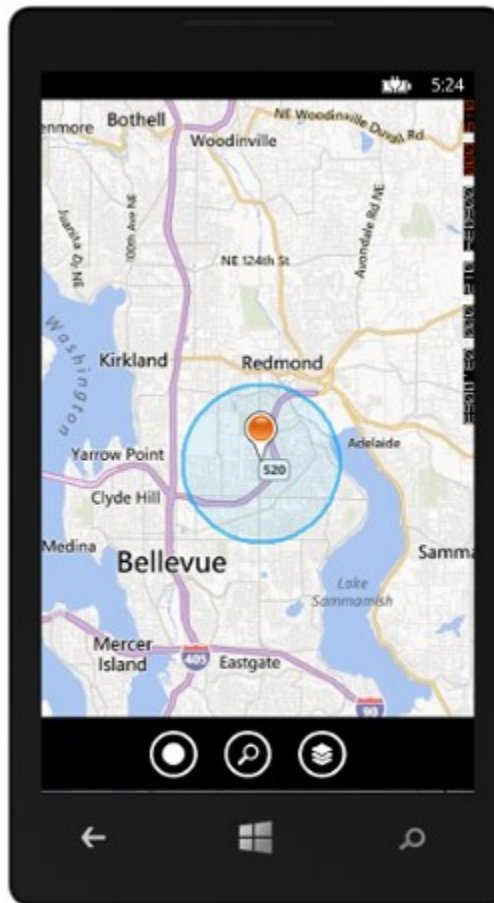
    Browser.Navigate(new Uri(MainUri, UriKind.Relative));
}
```

Visual Basic

```
Public Function Browser_Loaded(sender As Object, e As RoutedEventArgs)
    Browser.IsScriptEnabled = True
    Browser.IsGeolocationEnabled = True

    Browser.Navigate(New Uri(MainUri, UriKind.Relative))
End Function
```

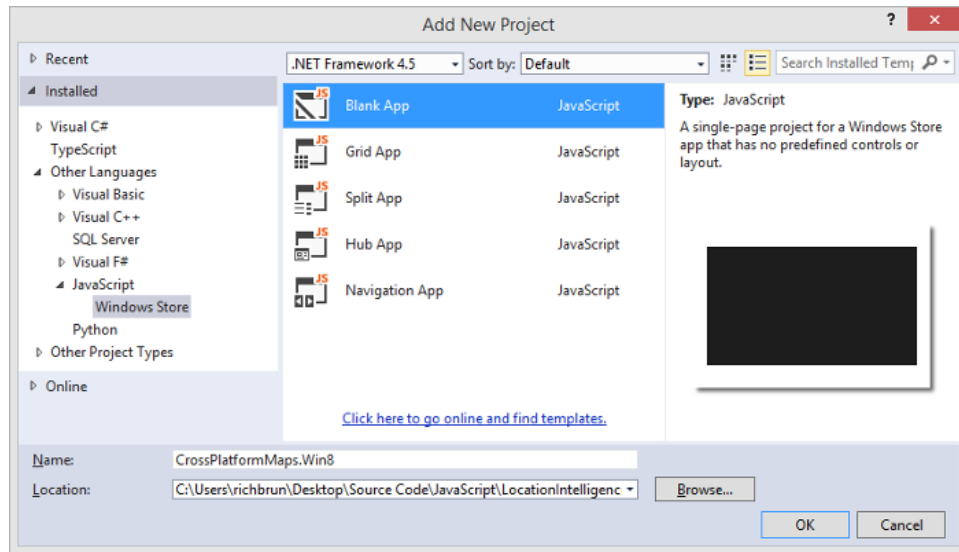
At this point the Windows Phone 8 version of this app is complete. Here is a screenshot of the app in the Windows Phone emulator.



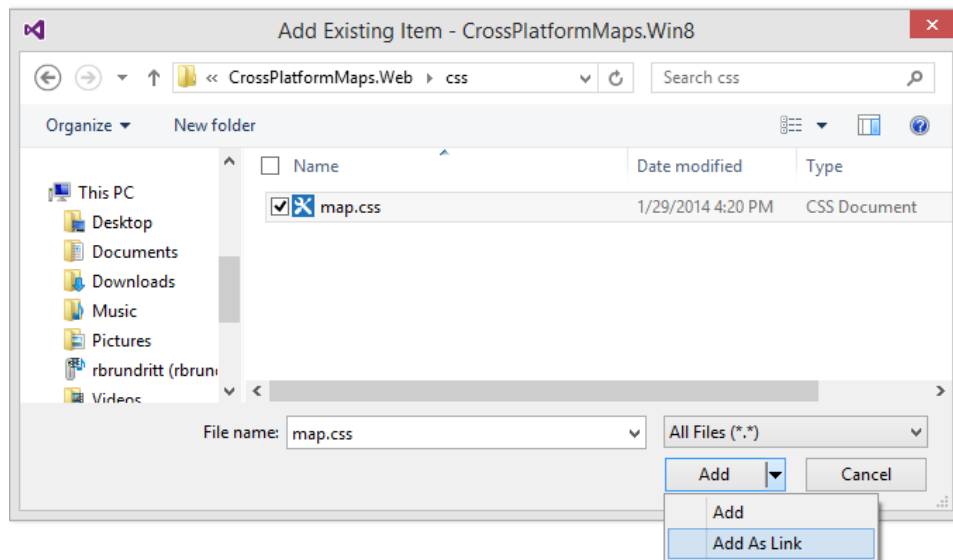
Integrating into a Windows Store App

As we already know Windows Store Apps can be created using JavaScript. We could simply use all the HTML and JavaScript files from our mobile friendly website as-is in our app and it would work fine. However, our mobile friendly website references the Bing Maps V7 AJAX control rather than the Windows Store version of Bing Maps. With a couple of minor changes we can get this app to use the Bing Maps Windows Store SDK. The benefit of doing this is that the map control will be hosted within the app which means it will be byte-code cached and load a lot faster.

To get started create a new project by right clicking on the solution and selecting **Add -> New Project**. From the JavaScript Windows Store templates section select the **Blank App** template and set the name to **CrossPlatformMaps.Win8**.



Next we will link in the CSS, JavaScript and image files from the web app into the **css**, **js** and **images** folder of the project. To do this right click on each folder and select **Add -> Existing Item**. Navigate to the **CrossPlatformMaps.Web** project and select the files in the respected folder. Instead of pressing the **Add** button, click on the arrow beside the **Add** button and select **Add As Link**. This will make the project reference the existing file from the other project. This makes things easier to manage as changing the file in either project will automatically update it in the other project.



Do not reference in the **index.html** file. Instead open up the **default.html** file and update it with the following HTML.

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

  <!-- WinJS references -->
  <link href="//Microsoft.WinJS.2.0/css/ui-dark.css" rel="stylesheet" />
  <script src="//Microsoft.WinJS.2.0/js/base.js"></script>
```



```

<script src="//Microsoft.WinJS.2.0/js/ui.js"></script>

<!-- BingMaps_W8_HTML5 references -->
<link href="/css/default.css" rel="stylesheet" />
<script src="/js/default.js"></script>

<!-- Bing Maps references -->
<script type="text/javascript" src="ms-
appx:///Bing.Maps.JavaScript/js/veapicore.js"></script>
<script type="text/javascript" src="ms-
appx:///Bing.Maps.JavaScript/js/veapimodules.js"></script>

<link rel="stylesheet" type="text/css" href="css/map.css" />

<script type="text/javascript" src="js/map.js"></script>
</head>
<body>
<div id="mapContainer">
<div id='myMap'></div>
<div class="appBar">
<div class="appBarContainer">



</div>
</div>
<div id="navBar">




</div>
<div id="zoomRotateBar">


<div id="rotationBtns">


</div>
</div>
</div>

<div id="searchPanel" style="display:none;">
<div>
<h2>Search</h2>
<input type="text" id="searchTbx" />

</div>
</div>

<div id="mapModePanel" style="display:none;">
<div>
<h2>Map Mode</h2>

<ul>
<li>
<input type="radio" name="mapMode" id="autoMode" value="auto"
checked="checked" />
<label for="autoMode">Auto</label>
</li>
<li>

```

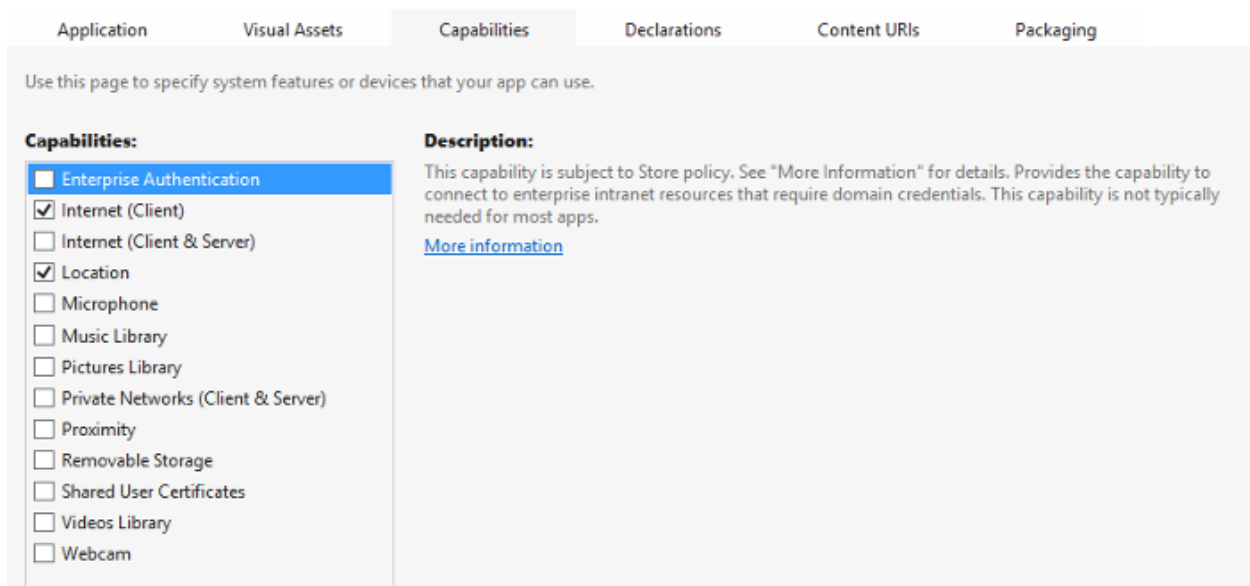
```

        <input type="radio" name="mapMode" id="aerialMode" value="aerial" />
        <label for="aerialMode">Aerial</label>
    </li>
    <li>
        <input type="radio" name="mapMode" id="birdseyeMode" value="birdseye" />
        <label for="birdseyeMode">Birdseye</label>
    </li>
    <li>
        <input type="radio" name="mapMode" id="roadMode" value="road" />
        <label for="roadMode">Road</label>
    </li>
</ul>
</div>
</div>
</body>
</html>

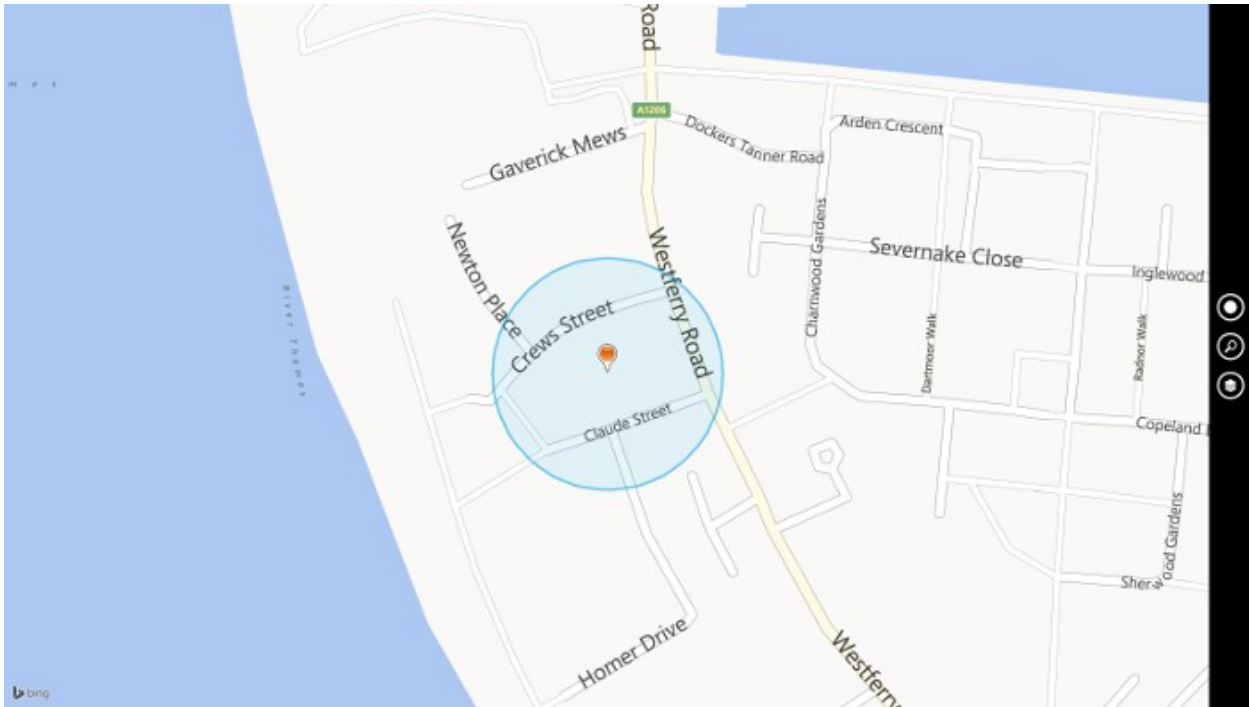
```

The main difference with this HTML and that which is in the **index.html** file is the added references to the WinJS libraries along with the Bing Maps Windows Store SDK. Also the workaround for desktop versions of Internet Explorer is removed as it is not needed.

Since our app will need access to the user's location to show it on the map we have to set this capability in the app. To do this, double click on the **package.appmanifest** file and select the **Capabilities** tab at the top. Under the **Capabilities** section, check the Location option.



At this point the Windows Store version of this app is complete. Here is a screenshot of the app showing my location.



jQuery Mobile in Apps

jQuery (<http://jquery.com/>) is a very popular JavaScript framework that makes it easier to develop JavaScript that works across different browsers. One branch of jQuery is called jQuery UI (<http://jqueryui.com/>) and includes a number of useful controls such as sliders, date picker, progress bar and to name a few. Another branch of jQuery is called jQuery Mobile (<http://jquerymobile.com/>) and includes a number of controls that are optimized for touch and look more like the controls found in native mobile apps. If you are creating a cross platform app using JavaScript this is definitely a library worth looking into.

Tip: As it was mentioned earlier in this book, you will need to use version 2 or above of jQuery in Windows Store apps as older versions throw security errors.

Wrap your apps with PhoneGap

PhoneGap (<http://phonegap.com/>) is a framework that allows you to create more powerful mobile apps using JavaScript and HTML. Typically when developing mobile apps using JavaScript we only have access to the resources that are available through the web browser control. Many native features, such as sensors, are not accessible through the web browser control. PhoneGap bridges this gap by providing a hybrid app template that exposes these features to the web browser by providing a combination of native and JavaScript libraries. Johannes Kebeck from the Bing Maps team has written a great blog post on using Bing Maps with PhoneGap to create cross platform applications. You can find the blog post here: <http://binged.it/1ezRP1C>

One common road block for a lot of developers when creating cross platform apps is not having a development environment setup for each mobile platform. To help with this there is an online service called PhoneGap Build which allows you to upload your HTML, JavaScript and CSS files to a server and it will compile it against all the different mobile platforms you want to target. This is a paid for service, however the first app is free and even if you build multiple apps the cost of the service is likely cheaper than what it would cost in time and hardware to create your development environment.

Creating a Cross Platform app using .NET

With each new version and branch of .NET there are a lot of differences. A method you might use in a Windows Store app may not be available or may be used differently in Windows Phone or WPF. When trying to share code between platforms developers often find that these differences only occur in a small amount of their code. Copying and pasting the code into another project and modifying the code to accommodate the change is not ideal as it often means that there is twice as much code to manage and if you make changes in one class you often have to make those changes in the other as well. Wouldn't it be great if we could simply wrap a block of code and indicate if it should be used when compiled against one platform or another?

Visual Studio's makes this possible with conditional compilation symbols. Conditional compilation symbols are "if" statements that are interpreted by the compiler in Visual Studios at compile time. These symbols allow us to specify which blocks of code to compile based on which symbols are defined in the project. One of the most common conditional compilation symbol used is **DEBUG**, which allows us to specify a block of code that will be compiled when Visual Studios is in debug mode. In addition to the **DEBUG** conditional compilation symbol we can also create custom symbols as we will see later in this chapter. Here is an example of how these symbols can be used.

C#

```
#if DEBUG
    //Code block to use when in debug mode.
#else
    //Code block to use when not in debug mode.
#endif
```

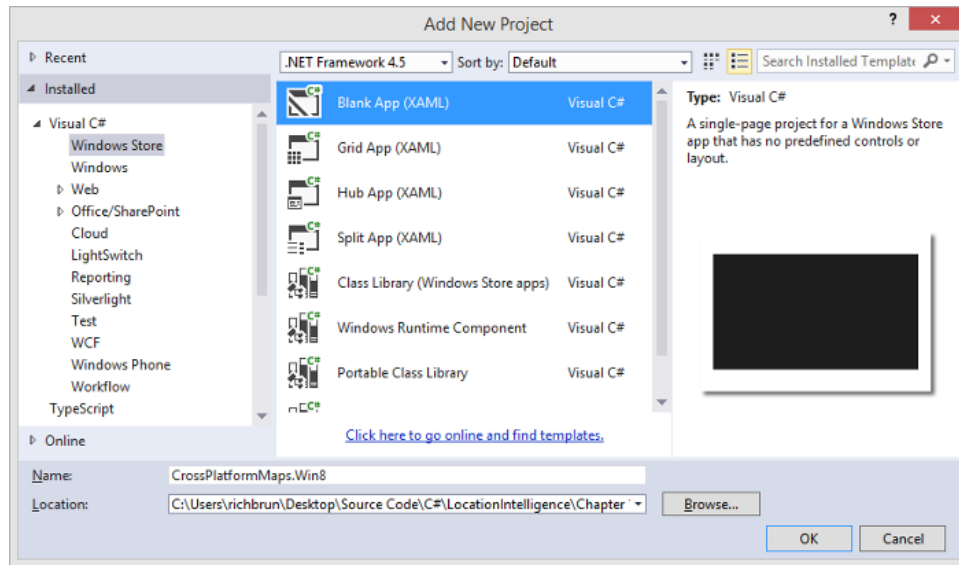
Visual Basic

```
#If DEBUG Then
    'Code block to use when in debug mode.
#Else
    'Code block to use when not in debug mode.
#End If
```

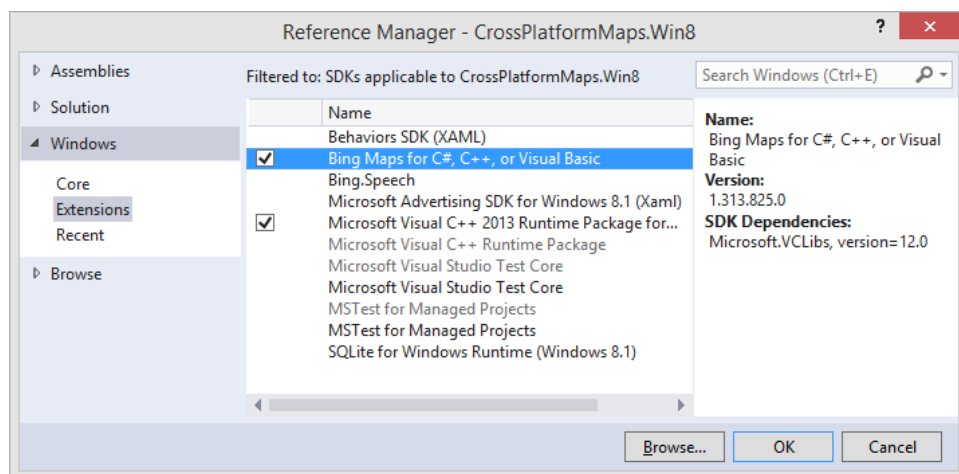
In this section of the chapter we are going to create two apps, a Windows Store and a Windows Phone app. As stated at the beginning of this chapter we are going to build a simple map app that allows the user to display their location and search for other locations. These apps will use conditional compilation symbols to reuse as much code as possible between the two apps. The maps in Windows Phone are different from those used in a Windows Store app. You can find documentation on the maps API in Windows Phone here: <http://bit.ly/1dSf99L>

Creating the base Project

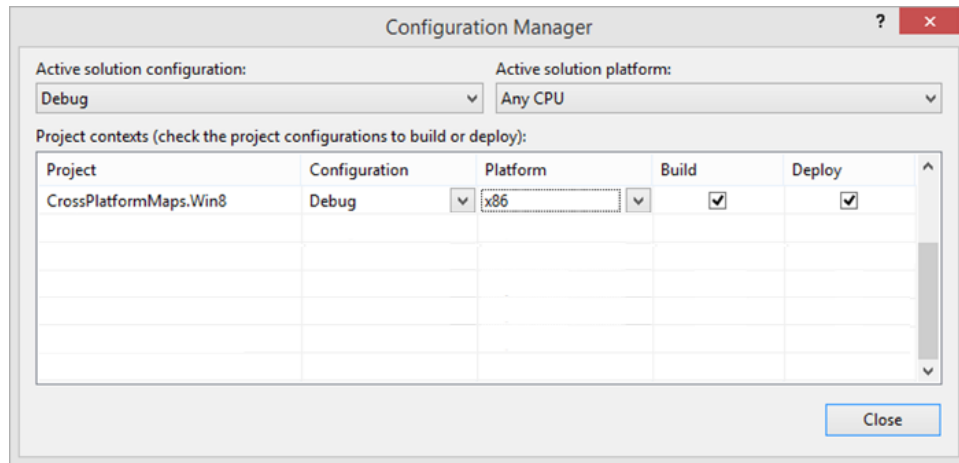
To get started, open Visual Studios and create a new Windows Store project in your preferred language; C# or Visual Basic. Select the **Blank App** template and call the application **CrossPlatformMaps.Win8** and press **OK**.



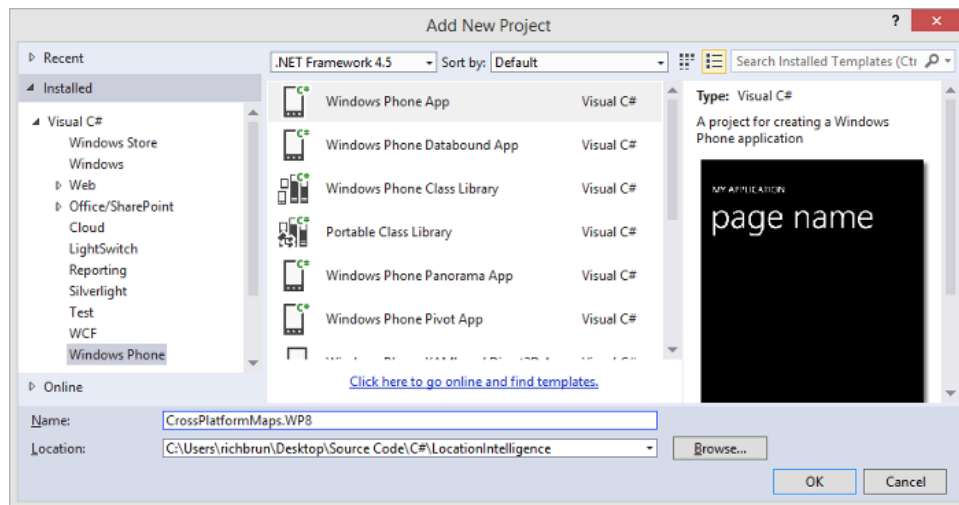
Add a reference to the Bing Maps SDK. To do this, right click on the **References** folder and press **Add Reference**. Select **Windows** -> **Extensions** select **Bing Maps for C#, C++ and Visual Basic** or **Bing Maps for JavaScript**. If you do not see this option ensure that you have installed the Bing Maps SDK for Windows Store apps. Also add a reference to the **Microsoft Visual C++ Runtime Package** as this is required by the Bing Maps SDK.



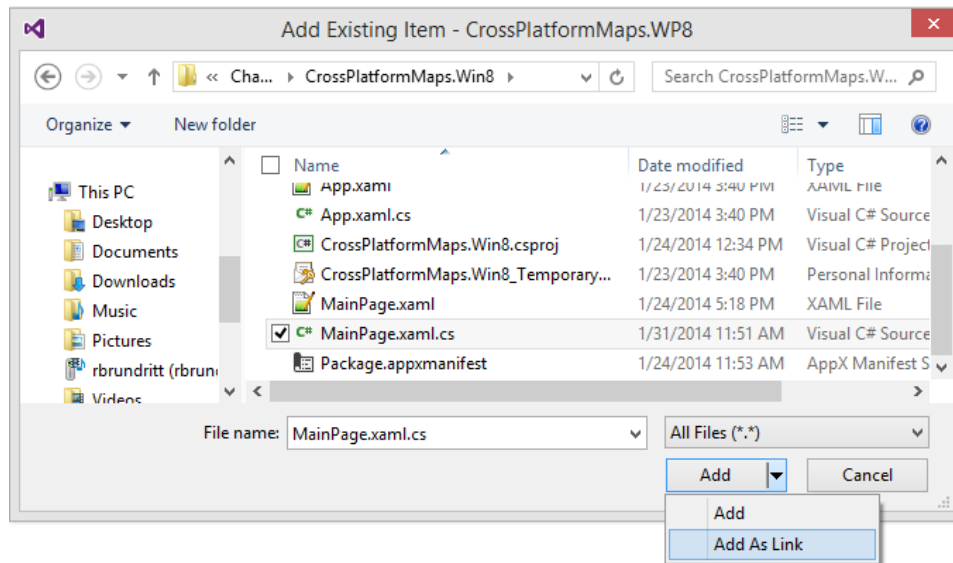
If you notice that there is a little yellow indicator on the references that you just added. The reason for this is that in the C++ runtime package you have to set the **Active solution platform** in Visual Studio to one of the following options; **ARM**, **x86** or **x64**. To do this, right click on the Solution folder and select **Properties**. Then go to **Configuration Properties** -> **Configuration**. Find your project and under the **Platform** column set the target platform to x86. Press OK and the yellow indicator should disappear from our references.



Next right click on the solution and select **Add -> New Project**. Create a new Windows Phone project that uses the **Windows Phone App** template and call the application **CrossPlatformMaps.WP8** and press **OK**.

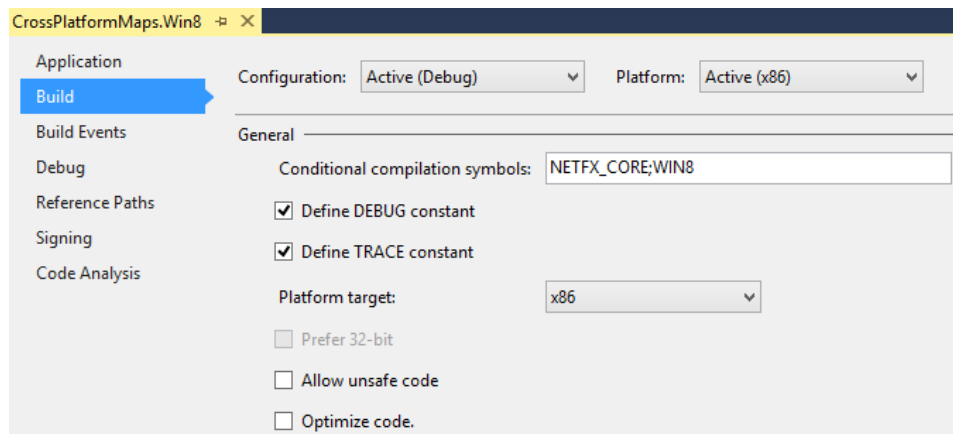


Locate the **MainPage.xaml.cs** file in this project and delete it. Right click on the project and select **Add -> Existing Item**. Navigate to the **CrossPlatformMaps.Win8** project and select the **MainPage.xaml.cs** file. Instead of pressing the **Add** button, click on the arrow beside the **Add** button and select **Add As Link**. This will make the project reference the existing file from the other project. This makes things easier to manage as changing the file in either project will automatically update it in the other project.

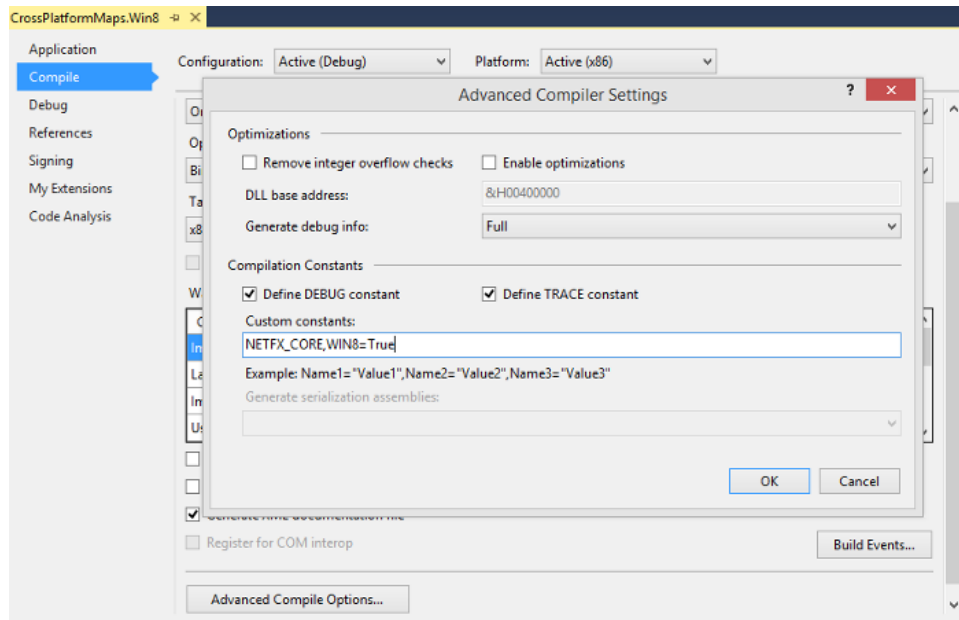


Defining Conditional Compilation Symbols

The next step is to define custom compilation symbols in the projects. The first symbol we will create will be in the Windows Store app and will be defined as **WIN8**. Right click on the **CrossPlatformMaps.Win8** project and select properties. If using C# click on the Build tab. Under the General section you will see a textbox for conditional compilation symbols. Add **WIN8** to this textbox.



If using Visual Basic click on the Compile tab and press the advance compile options button. In the custom constants textbox add **WIN8=True** and press OK.



Next for the Windows Phone project, right click on the **CrossPlatformMaps.WP8** project and select properties. If using C# click on the Build tab. Under the section labeled General you will see a textbox for conditional compilation symbols. Add **WINDOWS_PHONE** to this textbox. If using Visual Basic click on the Compile tab and press the advance compile options button. In the custom constants textbox add **WINDOWS_PHONE=True** and press OK.

Creating the User Interface

We are creating two apps that will be able to do essentially the same thing but on different platforms. As such it might not make sense for both apps to have the same user interface. In the Windows Store app it might make sense to display the search box over the map for easy access and to put the GPS button in the bottom app bar. Where as in the Windows Phone app we will usually have a lot less screen space so we will want to hide the search box until it's needed and will add all our buttons to the bottom app bar. Also, the maps in Windows Phone do not have a navigation bar for changing the map type, so we will add an option for this in the app.

In the Windows Store app we will create our buttons using Segoe UI Symbols like we have been doing throughout this book. In the Windows Phone app we can only use images to for app bar buttons. Add the following images to the **Assets** folder of the **CrossPlatformMaps.WP8** project.



Next open the **MainPage.xaml** file in the **CrossPlatformMaps.Win8** project and update it with the following XAML.

```
<Page
  x:Class="CrossPlatformMaps.Win8.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:CrossPlatformMaps.Win8"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:m="using:Bing.Maps">
```



```

<Page.BottomAppBar>
    <AppBar>
        <StackPanel Orientation="Horizontal">
            <AppBarButton Label="GPS" Tapped="GpsBtn_Tapped">
                <AppBarButton.Icon>
                    <FontIcon Glyph="Ⓜ"/>
                </AppBarButton.Icon>
            </AppBarButton>
        </StackPanel>
    </AppBar>
</Page.BottomAppBar>

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <m:Map Name="MyMap" Credentials="YOUR_BING_MAPS_KEY"/>

    <StackPanel Orientation="Horizontal" Margin="0,100,0,0" VerticalAlignment="Top"
HorizontalAlignment="Right">
        <TextBox Name="SearchTbx" Width="300" Margin="0,0,10,0"/>
        <Button Content="Search" Tapped="GeocodeBtn_Tapped" Background="Gray"/>
    </StackPanel>
</Grid>
</Page>

```

Next go to the **CrossPlatformMaps.WP8** project and open the **MainPage.xaml** file and update it with the following XAML.

```

<phone:PhoneApplicationPage
    x:Class="CrossPlatformMaps.WP8.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:m="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="PortraitOrLandscape" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">

    <phone:PhoneApplicationPage.ApplicationBar>
        <shell:ApplicationBar IsVisible="True">
            <shell:ApplicationBarIconButton Text="GPS" IconUri="/Assets/gps.png"
Click="GpsBtn_Tapped"/>
            <shell:ApplicationBarIconButton Text="Search" IconUri="/Assets/search.png"
Click="SearchBtn_Tapped"/>
            <shell:ApplicationBarIconButton Text="Map Mode" IconUri="/Assets/layers.png"
Click="MapModeBtn_Tapped"/>
        </shell:ApplicationBar>
    </phone:PhoneApplicationPage.ApplicationBar>

    <Grid x:Name="LayoutRoot" Background="Transparent">
        <m:Map Name="MyMap" />

        <Grid Name="SearchPanel" Visibility="Collapsed">
            <Border Background="Black" Opacity="0.8"/>
            <StackPanel Orientation="Horizontal" VerticalAlignment="Center"
HorizontalAlignment="Center">
                <TextBox Name="SearchTbx" MinWidth="300"/>
                <Button Content="Search" Click="GeocodeBtn_Tapped"/>
            </StackPanel>
        </Grid>
    </Grid>

```

```

        </StackPanel>
    </Grid>

    <Grid Name="MapModePanel" Visibility="Collapsed">
        <Border Background="Black" Opacity="0.8"/>
        <StackPanel>
            <RadioButton Content="Road" Tag="Road" Click="MapMode_Selected"
IsChecked="True"/>
            <RadioButton Content="Aerial" Tag="Aerial" Click="MapMode_Selected"/>
            <RadioButton Content="Hybrid" Tag="Hybrid" Click="MapMode_Selected"/>
            <RadioButton Content="Terrain" Tag="Terrain" Click="MapMode_Selected"/>
        </StackPanel>
    </Grid>
</phone:PhoneApplicationPage>

```

Adding the application logic

Now that we have the base projects setup and the user interface created we can now add the code to power the app. Open the **MainPage.xaml.cs** file, it doesn't matter which project you do this in as it is linked between the two of them. The first thing we need to do is reference in the different libraries we will need, and add the appropriate namespace. When the app loads we will add a **MapLayer** to the map. We will use this layer to add pushpins to the map. Update the **MainPage.xaml.cs** file with the following code.

C#

```

using System;

#if WIN8

using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Input;
using Windows.Devices.Geolocation;
using Bing.Maps;
using Windows.UI.Xaml;
using Bing.Maps.Search;

namespace CrossPlatformMaps.Win8
{
    public sealed partial class MainPage : Page
    {
#elif WINDOWS_PHONE

using Microsoft.Phone.Controls;
using Windows.Devices.Geolocation;
using System.Device.Location;
using Microsoft.Phone.Maps.Services;
using Microsoft.Phone.Maps.Controls;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Shapes;
using System.Windows.Media;

namespace CrossPlatformMaps.WP8
{
    public partial class MainPage : PhoneApplicationPage
    {
#endif
        private MapLayer pinLayer;

```

```

public MainPage()
{
    this.InitializeComponent();

    MyMap.Loaded += (s, e) =>
    {
        //Create a map layer for displaying the pins on
        pinLayer = new MapLayer();

        #if WIN8
        MyMap.Children.Add(pinLayer);
        #elif WINDOWS_PHONE
        MyMap.Layers.Add(pinLayer);
        #endif
    };
}
}
}

```

Visual Basic

```

#If WIN8 Then

Imports Windows.UI.Xaml.Controls
Imports Windows.UI.Xaml.Input
Imports Windows.Devices.Geolocation
Imports Bing.Maps
Imports Windows.UI.Xaml
Imports Bing.Maps.Search

Public NotInheritable Class MainPage
    Inherits Page

#ElseIf WINDOWS_PHONE Then

Imports Microsoft.Phone.Controls
Imports Windows.Devices.Geolocation
Imports System.Device.Location
Imports Microsoft.Phone.Maps.Services
Imports Microsoft.Phone.Maps.Controls
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Shapes
Imports System.Windows.Media

Partial Public Class MainPage
    Inherits PhoneApplicationPage

#End If

    Dim pinLayer As MapLayer

    Public Sub New()
        InitializeComponent()

        AddHandler MyMap.Loaded,
            Sub(s, e)
                'Create a map layer for displaying the pins on
                pinLayer = New MapLayer()
            End Sub
    End Sub

```

```

#If WIN8 Then
    MyMap.Children.Add(pinLayer)
#ElseIf WINDOWS_PHONE Then
    MyMap.Layers.Add(pinLayer)
#End If
    End Sub
End Sub
End Class

```

Next we will add the event handler for when the user clicks the GPS button. Button event handlers in Windows Store apps take in a different set of parameters than the equivalent event handlers in Windows Phone. To handle this we can wrap just the first line of the event handler with conditional compilation symbols so that we can still reuse the code inside the handler. The maps in Windows Phone use a class called **GeoCoordinate** instead of a **Location** class like we use in Bing Maps for Windows Store apps. Add the following event handler to **MainPage.xaml.cs** file.

C#

```

#if WIN8
private async void GpsBtn_Tapped(object sender, TappedRoutedEventArgs e)
#elseif WINDOWS_PHONE
private async void GpsBtn_Tapped(object sender, System.EventArgs e)
#endif
{
    try
    {
        Geolocator geolocator = new Geolocator();
        geolocator.DesiredAccuracy = PositionAccuracy.High;

        Geoposition currentPosition = await
        geolocator.GetGeopositionAsync(TimeSpan.FromMinutes(1),
        TimeSpan.FromSeconds(10));
        #if WIN8
        await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal, () =>
        {
            MyMap.Center = new
            Location(currentPosition.Coordinate.Point.Position.Latitude,
            currentPosition.Coordinate.Point.Position.Longitude);
        });
        #elseif WINDOWS_PHONE
        Dispatcher.BeginInvoke(() =>
        {
            MyMap.Center = new GeoCoordinate(currentPosition.Coordinate.Latitude,
            currentPosition.Coordinate.Longitude);
        });
        #endif
    }
    catch { }
}

```

Visual Basic

```

#If WIN8 Then
Private Async Sub GpsBtn_Tapped(sender As Object, e As TappedRoutedEventArgs)
#Elseif WINDOWS_PHONE Then
Private Async Sub GpsBtn_Tapped(sender As Object, e As EventArgs)
#End If
Try

```

```

Dim geolocator = New Geolocator()
geolocator.DesiredAccuracy = PositionAccuracy.High

Dim currentPosition = Await
geolocator.GetGeopositionAsync(TimeSpan.FromMinutes(1), TimeSpan.FromSeconds(10))

#If WIN8 Then
    Await Dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.Normal,
        Sub()
            MyMap.Center = New
Location(currentPosition.Coordinate.Point.Position.Latitude,
currentPosition.Coordinate.Point.Position.Longitude)
        End Sub)
#ElseIf WINDOWS_PHONE Then
    Dispatcher.BeginInvoke(Sub()
        MyMap.Center = New
System.Device.Location.GeoCoordinate(currentPosition.Coordinate.Latitude,
currentPosition.Coordinate.Longitude)
        End Sub)
#End If
Catch
End Try
End Sub

```

When the search button in the Windows Phone app is pressed we will want to display the search panel which has a textbox in it. When the geocoding button is pressed in either app we will want to take the text from the textbox and geocode it for the user. The app will zoom into the first result and display it on the map with a pushpin. In this case the code for the Windows Store app has a lot of differences from the Windows Phone app and it's a lot easier and cleaner to have separate methods for each platform. Add the following code to **MainPage.xaml.cs** file.

C#

```

#if WIN8
private async void GeocodeBtn_Tapped(object sender, TappedRoutedEventArgs e)
{
    try
    {
        if (!string.IsNullOrWhiteSpace(SearchTbx.Text))
        {
            var options = new GeocodeRequestOptions(SearchTbx.Text);
            var response = await MyMap.SearchManager.GeocodeAsync(options);

            if (response.LocationData != null &&
                response.LocationData.Count > 0)
            {
                //Add pushpin to show geocoded location
                Pushpin pin = new Pushpin();
                MapLayer.SetPosition(pin, response.LocationData[0].Location);
                pinLayer.Children.Add(pin);

                //Position the map over the location
                MyMap.SetView(response.LocationData[0].Location, 15);
            }
        }
    }
    catch { }
}
#elseif WINDOWS_PHONE
private void SearchBtn_Tapped(object sender, System.EventArgs e)
{

```

```

        SearchPanel.Visibility = Visibility.Visible;
    }

    private void GeocodeBtn_Tapped(object sender, System.Windows.RoutedEventArgs e)
    {
        if (!string.IsNullOrEmpty(SearchTbx.Text))
        {
            var query = new GeocodeQuery();
            query.SearchTerm = SearchTbx.Text;
            query.GeoCoordinate = MyMap.Center;
            query.QueryCompleted += (s, a) =>
            {
                pinLayer.Clear();

                if (a.Result != null && a.Result.Count > 0)
                {
                    var pin = new MapOverlay();
                    pin.Content = new Ellipse()
                    {
                        Fill = new SolidColorBrush(Colors.Blue),
                        Stroke = new SolidColorBrush(Colors.White),
                        StrokeThickness = 5,
                        Width = 30,
                        Height = 30
                    };
                    pin.GeoCoordinate = a.Result[0].GeoCoordinate;
                    pinLayer.Add(pin);

                    MyMap.SetView(a.Result[0].GeoCoordinate, 15);
                }

                SearchPanel.Visibility = Visibility.Collapsed;
            };
            query.QueryAsync();
        }
    }
}
#endif

```

Visual Basic

```

#If WIN8 Then
Private Async Sub GeocodeBtn_Tapped(sender As Object, e As TappedRoutedEventArgs)
    Try
        If (Not String.IsNullOrEmpty(SearchTbx.Text)) Then
            Dim options = New GeocodeRequestOptions(SearchTbx.Text)
            Dim response = Await MyMap.SearchManager.GeocodeAsync(options)

            If (response.LocationData IsNot Nothing And response.LocationData.Count > 0)
Then
                'Add pushpin to show geocoded location
                Dim pin = New Pushpin()
                MapLayer.SetPosition(pin, response.LocationData(0).Location)
                pinLayer.Children.Add(pin)

                'Position the map over the location
                MyMap.SetView(response.LocationData(0).Location, 15)
            End If
        End If
    Catch
    End Try

```

```

End Sub#ElseIf WINDOWS_PHONE Then
Private Sub SearchBtn_Tapped(sender As Object, e As EventArgs)
    SearchPanel.Visibility = Visibility.Visible
End Sub

Private Sub GeocodeBtn_Tapped(sender As Object, e As RoutedEventArgs)
    If (Not String.IsNullOrEmpty(SearchTbx.Text)) Then

        Dim query = New GeocodeQuery()
        query.SearchTerm = SearchTbx.Text
        query.GeoCoordinate = MyMap.Center

        AddHandler query.QueryCompleted,
            Sub(s, a)
                pinLayer.Clear()

                If (a.Result IsNot Nothing And a.Result.Count > 0) Then
                    Dim pinContent = New Ellipse()
                    pinContent.Fill = New SolidColorBrush(Colors.Blue)
                    pinContent.Stroke = New SolidColorBrush(Colors.White)
                    pinContent.StrokeThickness = 5
                    pinContent.Width = 30
                    pinContent.Height = 30

                    Dim pin = New MapOverlay()
                    pin.Content = pinContent

                    pin.GeoCoordinate = a.Result(0).GeoCoordinate
                    pinLayer.Add(pin)

                    MyMap.SetView(a.Result(0).GeoCoordinate, 15)
                End If

                SearchPanel.Visibility = Visibility.Collapsed
            End Sub

        query.QueryAsync()
    End If
End Sub
#End If

```

The final bit of code left to add is the logic for changing the map type. This code is only needed for the Windows Phone app so we can wrap all the code with a single conditional compilation symbol. Add the following code to the **MainPage.xaml.cs** file.

C#

```

#if WINDOWS_PHONE
private void MapModeBtn_Tapped(object sender, System.EventArgs e)
{
    MapModePanel.Visibility = Visibility.Visible;
}

private void MapMode_Selected(object sender, RoutedEventArgs e)
{
    var btn = sender as RadioButton;

    switch (btn.Tag.ToString())
    {
        case "Road":

```

```

        MyMap.CartographicMode = MapCartographicMode.Road;
        break;
    case "Aerial":
        MyMap.CartographicMode = MapCartographicMode.Aerial;
        break;
    case "Hybrid":
        MyMap.CartographicMode = MapCartographicMode.Hybrid;
        break;
    case "Terrain":
        MyMap.CartographicMode = MapCartographicMode.Terrain;
        break;
    }

    MapModePanel.Visibility = Visibility.Collapsed;
}
#endif

```

Visual Basic

```

#if WINDOWS_PHONE Then
Private Sub MapModeBtn_Tapped(sender As Object, e As EventArgs)
    MapModePanel.Visibility = Visibility.Visible
End Sub

Private Sub MapMode_Selected(sender As Object, e As RoutedEventArgs)
    Dim btn = TryCast(sender, RadioButton)

    Select Case btn.Tag.ToString()
        Case "Road"
            MyMap.CartographicMode = MapCartographicMode.Road
            Exit Select
        Case "Aerial"
            MyMap.CartographicMode = MapCartographicMode.Aerial
            Exit Select
        Case "Hybrid"
            MyMap.CartographicMode = MapCartographicMode.Hybrid
            Exit Select
        Case "Terrain"
            MyMap.CartographicMode = MapCartographicMode.Terrain
            Exit Select
    End Select

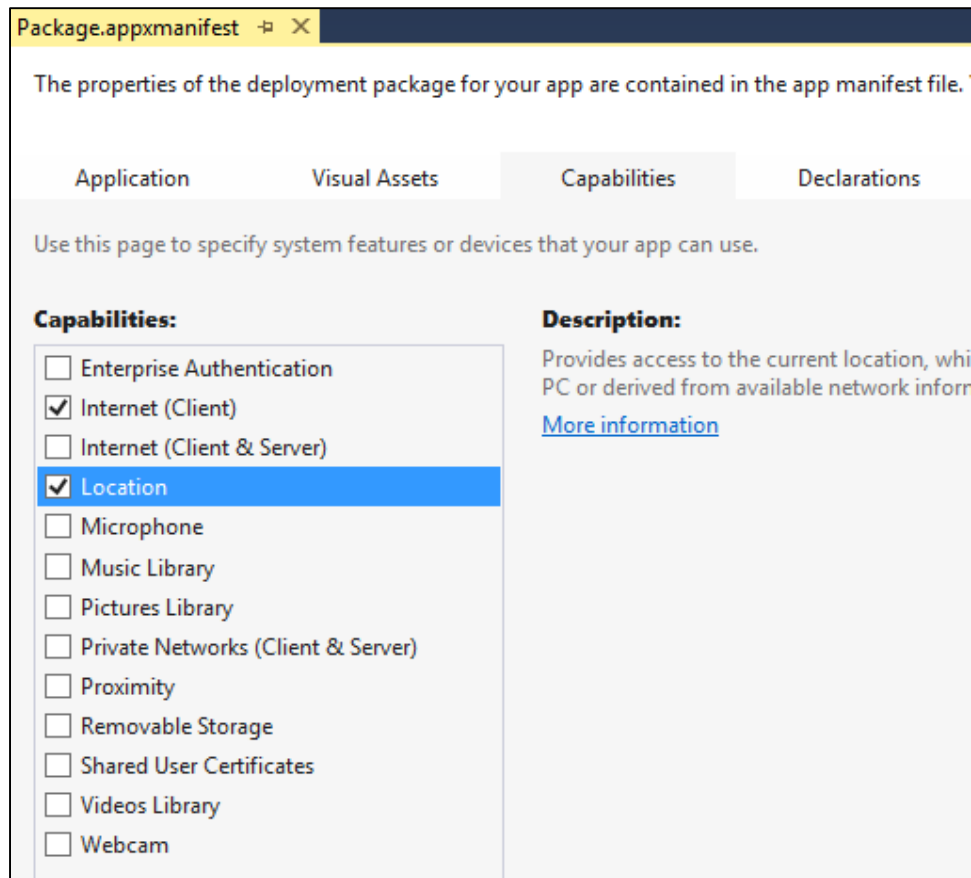
    MapModePanel.Visibility = Visibility.Collapsed
End Sub
#End If

```

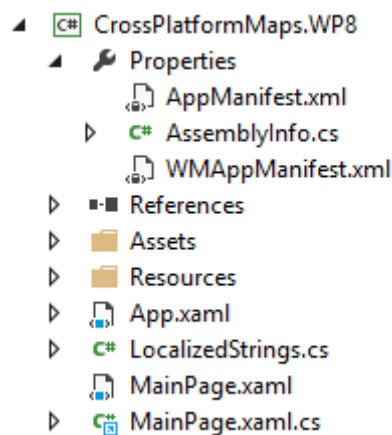
Enabling the Capabilities of the apps

At this point the apps are nearly complete. The only thing left is to enable the capabilities of the apps in the manifest. In the Windows Store app we will need to enable the location capability as our app will need to access the GPS. In the Windows Phone app we will need to enable both the location and the map capabilities.

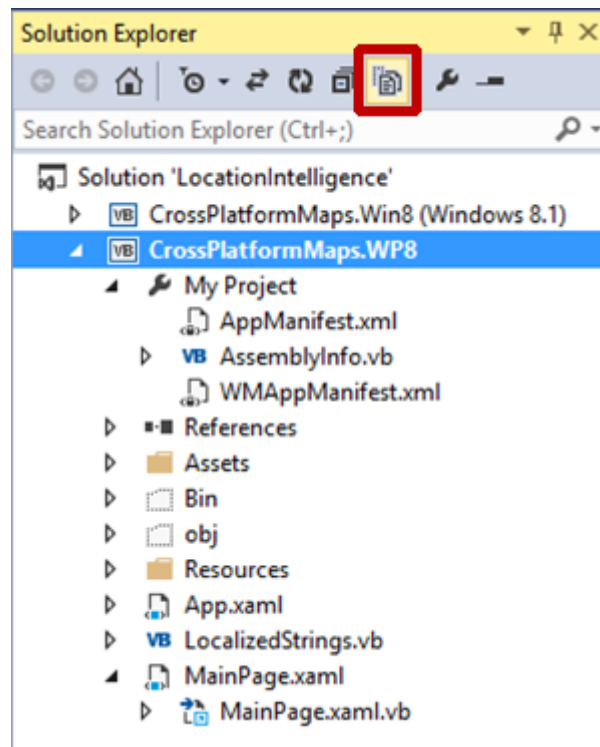
Let's start with the Windows Store project by double clicking on the **Package.appxmanifest** file and under the Capabilities tab check the Location option.



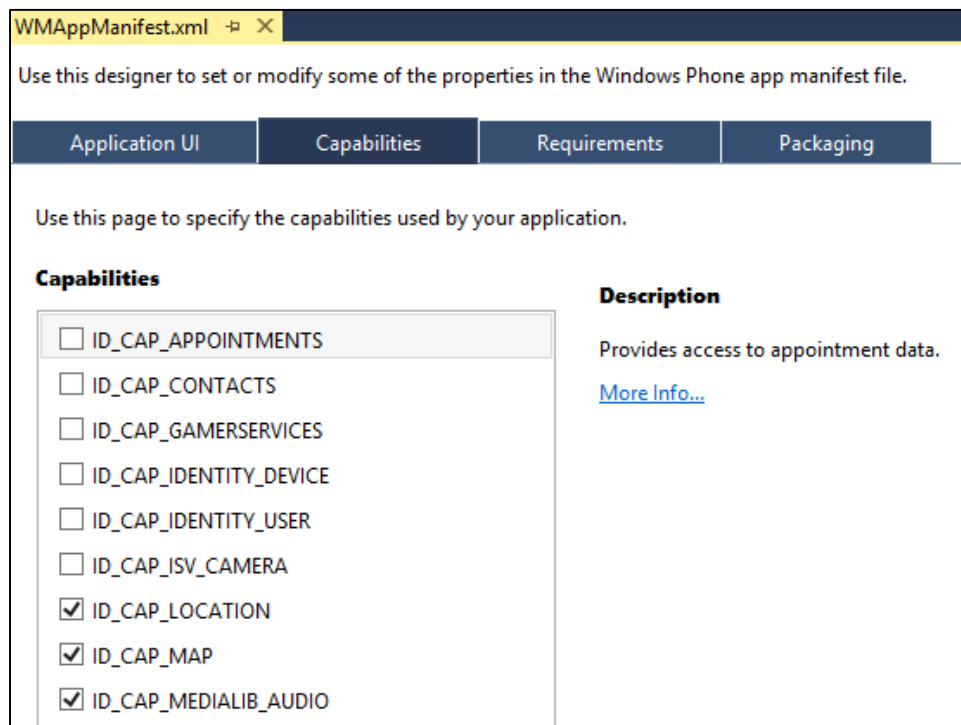
Next go to the Windows Phone app. If you are using C#, expand the Properties section of the project and double click on the **WMAAppManifest.xml** file.



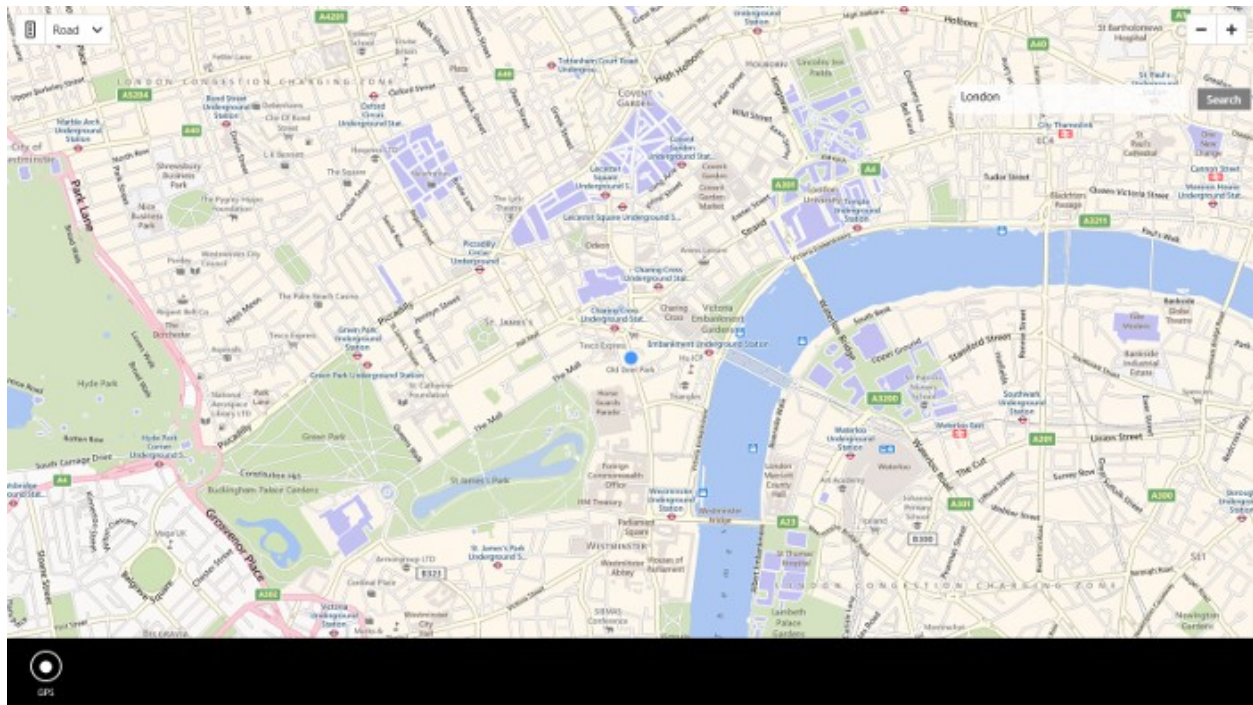
If you are using Visual Basic, press the Show All Files button in the Solution Explorer then expand the My Properties section of the project and double click on the **WMAAppManifest.xml** file.



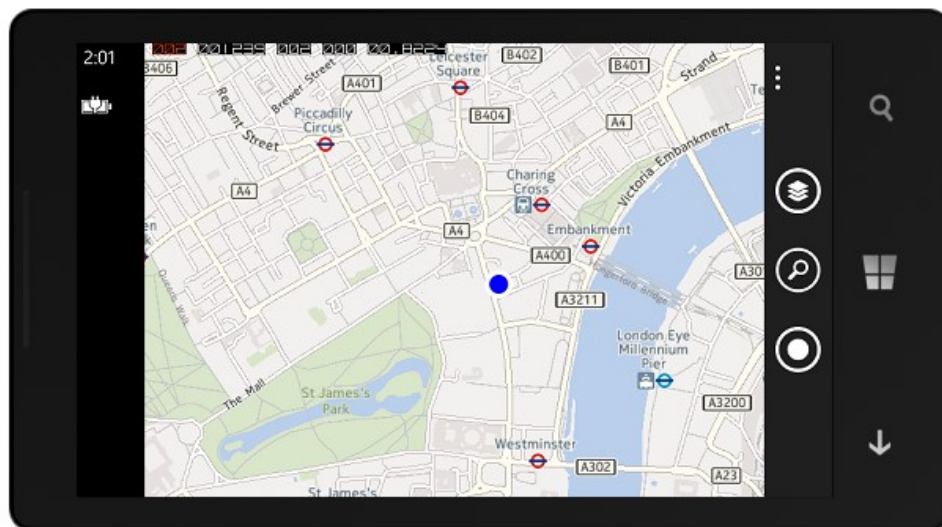
In the window that opens, select the Capabilities tab and select the ID_CAP_MAP and the ID_CAP_LOCATION options.



At this point both apps are complete and ready for testing. If you run the Windows Store app and search for London it should look like the following screenshot. Note I've opened the bottom app bar so you can see the GPS button.



If you run the Windows Phone app and do a search for London the app will look like the following image.

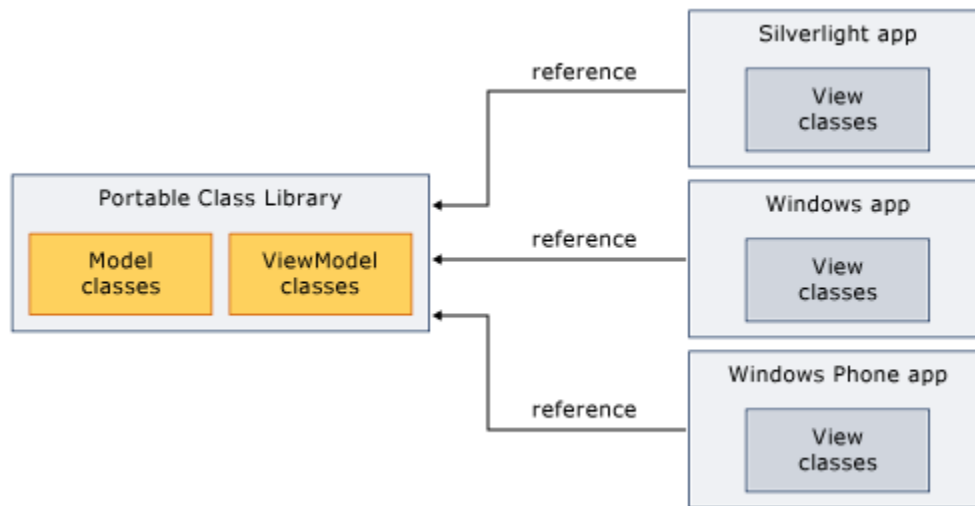


Portable Class Libraries in .NET

So far in this chapter we saw how to use conditional compilation symbols to help with code reuse when creating cross platform applications. Another great tool for creating cross platform apps in .NET is to create portable class libraries. The Portable Class Library project was introduced in Visual Studio 2012 and supports a subset of assemblies from the

.NET Framework, Silverlight, .NET for Windows Store apps, Windows Phone, and Xbox 360. This provides a Visual Studio template that you can use to build assemblies that run without modification on these platforms. If you don't use a portable class library project, you must target a single app type, and then manually rework the class library for other app types. With the portable class library project, you can reduce the time and costs of developing and testing code by building portable assemblies that are shared across apps for different devices.

One common pattern for creating cross platform apps is the Model-View-View-Model (MVVM) pattern. MVVM is an application pattern that isolates the user interface from the underlying business logic. You can implement the model and view model classes in a portable class library project, and then create the views separately for each platform. This approach enables you to write the data model and business logic only once, and use that code in .NET Framework, Silverlight, Windows Phone, and Windows Store apps, as shown in the following image.



There is great documentation on how to create cross platform apps using portable class libraries available here: <http://bit.ly/1cDSmM>

You can also find a video on Channel 9 about creating cross platform apps using portable class libraries here: <http://bit.ly/1bdwOfk>

Chapter Summary

In this chapter you were introduced to two different methods for creating cross platform apps. Here is a short summary of some key points from this chapter.

- Besides using JavaScript and HTML to create Windows Store and Windows Phone apps you can also embed these types of apps into a web browser control on other platforms such as iOS, Android, and Blackberry.
- The Bing Maps V7 AJAX control is supported on all major PC and Mac browsers, Windows Phone 8, iOS, Android, and Blackberry 6 and above. Also, since Kindle Fire is essentially a heavily modified version of Android, it works on their too.
- Rather than writing code that targets specific browsers or operating systems, write code that tests for features in JavaScript instead. This will increase the chances of your code continuing to work in future versions of web browsers.
- In Internet Explorer 11 you can easily emulate different browsers to see how your app works in them. Simply press F12 when Internet Explorer is open and it will display the developer's tools to you.
- jQuery is a great JavaScript library that takes a lot of the pain out of creating JavaScript apps that work across different browsers. Version 2 of jQuery should be used in Windows Store apps. jQuery Mobile

(<http://jquerymobile.com/>) includes a number of controls that are optimized for touch and look more like the controls found in native mobile apps

- PhoneGap is a framework that lets you wrap your HTML and JavaScript apps in native wrappers for all the major mobile platforms. It also exposes many features to JavaScript that are normally only available to native apps by using a hybrid approach.
- Conditional compilation symbols are “if” statements that are interpreted by the compiler in Visual Studios at compile time. These symbols allow us to specify which blocks of code to compile based on which symbols are defined in the project.
- The Portable Class Library project was introduced in Visual Studio 2012 and supports a subset of assemblies from the .NET Framework, Silverlight, .NET for Windows Store apps, Windows Phone, and Xbox 360. This provides a Visual Studio template that you can use to build assemblies that run without modification on these platforms. There is great documentation on how to create cross platform apps using portable class libraries available here: <http://bit.ly/1cDSmM>
- Model-View-View-Model (MVVM) is an application pattern that isolates the user interface from the underlying business logic. You can implement the model and view model classes in a portable class library project, and then create the views separately for each platform.

Appendix A: ISO-3166 Country Codes

ISO-3166 is a standard published by the International Organization for Standardization (ISO). It defines codes for the names of countries as 2 letter, 3 letter and numeric codes. The 2 letter country code is the most widely used. The following is a list of all the ISO-3166 2 letter country codes.

Country Name	ISO Code
Afghanistan	AF
Aland Islands	AX
Albania	AL
Algeria	DZ
American Samoa	AS
Andorra	AD
Angola	AO
Anguilla	AI
Antarctica	AQ
Antigua and Barbuda	AG
Argentina	AR
Armenia	AM
Aruba	AW
Australia	AU
Austria	AT
Azerbaijan	AZ
Bahamas	BS
Bahrain	BH
Bangladesh	BD
Barbados	BB
Belarus	BY
Belgium	BE
Belize	BZ
Benin	BJ
Bermuda	BM
Bhutan	BT
Bolivia, Plurinational State of	BO
Bonaire, Sint Eustatius and Saba	BQ
Bosnia and Herzegovina	BA
Botswana	BW
Bouvet Island	BV
Brazil	BR
British Indian Ocean Territory	IO
Brunei Darussalam	BN
Bulgaria	BG
Burkina Faso	BF
Burundi	BI
Cambodia	KH
Cameroon	CM
Canada	CA
Cape Verde	CV
Cayman Islands	KY

Central African Republic	CF
Chad	TD
Chile	CL
China	CN
Christmas Island	CX
Cocos (Keeling) Islands	CC
Colombia	CO
Comoros	KM
Congo	CG
Congo, the Democratic Republic of the	CD
Cook Islands	CK
Costa Rica	CR
Cote d'Ivoire	CI
Croatia	HR
Cuba	CU
Curacao	CW
Cyprus	CY
Czech Republic	CZ
Denmark	DK
Djibouti	DJ
Dominica	DM
Dominican Republic	DO
Ecuador	EC
Egypt	EG
El Salvador	SV
Equatorial Guinea	GQ
Eritrea	ER
Estonia	EE
Ethiopia	ET
Falkland Islands (Malvinas)	FK
Faroe Islands	FO
Fiji	FJ
Finland	FI
France	FR
French Guiana	GF
French Polynesia	PF
French Southern Territories	TF
Gabon	GA
Gambia	GM
Georgia	GE
Germany	DE
Ghana	GH
Gibraltar	GI

Greece	GR
Greenland	GL
Grenada	GD
Guadeloupe	GP
Guam	GU
Guatemala	GT
Guernsey	GG
Guinea	GN
Guinea-Bissau	GW
Guyana	GY
Haiti	HT
Heard Island and McDonald Islands	HM
Holy See (Vatican City State)	VA
Honduras	HN
Hong Kong	HK
Hungary	HU
Iceland	IS
India	IN
Indonesia	ID
Iran, Islamic Republic of	IR
Iraq	IQ
Ireland	IE
Isle of Man	IM
Israel	IL
Italy	IT
Jamaica	JM
Japan	JP
Jersey	JE
Jordan	JO
Kazakhstan	KZ
Kenya	KE
Kiribati	KI
Korea, Democratic People's Republic of	KP
Korea, Republic of	KR
Kuwait	KW
Kyrgyzstan	KG
Lao People's Democratic Republic	LA
Latvia	LV
Lebanon	LB
Lesotho	LS
Liberia	LR
Libya	LY
Liechtenstein	LI
Lithuania	LT
Luxembourg	LU
Macao	MO
Macedonia, The Former Yugoslav Republic of	MK
Madagascar	MG
Malawi	MW
Malaysia	MY

Maldives	MV
Mali	ML
Malta	MT
Marshall Islands	MH
Martinique	MQ
Mauritania	MR
Mauritius	MU
Mayotte	YT
Mexico	MX
Micronesia, Federated States of	FM
Moldova, Republic of	MD
Monaco	MC
Mongolia	MN
Montenegro	ME
Montserrat	MS
Morocco	MA
Mozambique	MZ
Myanmar	MM
Namibia	NA
Nauru	NR
Nepal	NP
Netherlands	NL
New Caledonia	NC
New Zealand	NZ
Nicaragua	NI
Niger	NE
Nigeria	NG
Niue	NU
Norfolk Island	NF
Northern Mariana Islands	MP
Norway	NO
Oman	OM
Pakistan	PK
Palau	PW
Palestine, State of	PS
Panama	PA
Papua New Guinea	PG
Paraguay	PY
Peru	PE
Philippines	PH
Pitcairn	PN
Poland	PL
Portugal	PT
Puerto Rico	PR
Qatar	QA
Reunion	RE
Romania	RO
Russian Federation	RU
Rwanda	RW
Saint Barthelemy	BL

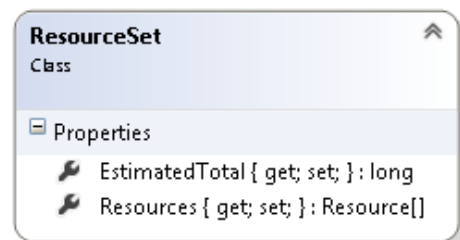
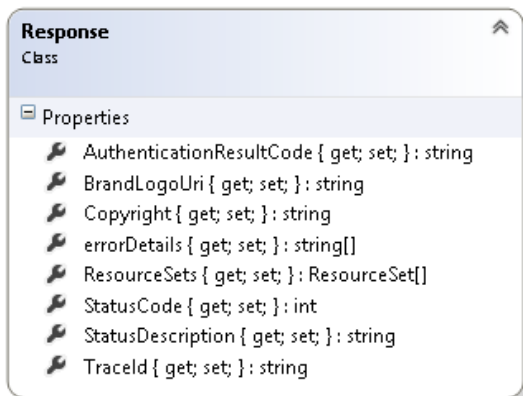
Saint Helena, Ascension and Tristan da Cunha	SH
Saint Kitts and Nevis	KN
Saint Lucia	LC
Saint Martin (French part)	MF
Saint Pierre and Miquelon	PM
Saint Vincent and the Grenadines	VC
Samoa	WS
San Marino	SM
Sao Tome and Principe	ST
Saudi Arabia	SA
Senegal	SN
Serbia	RS
Seychelles	SC
Sierra Leone	SL
Singapore	SG
Sint Maarten (Dutch part)	SX
Slovakia	SK
Slovenia	SI
Solomon Islands	SB
Somalia	SO
South Africa	ZA
South Georgia and the South Sandwich Islands	GS
South Sudan	SS
Spain	ES
Sri Lanka	LK
Sudan	SD
Suriname	SR
Svalbard and Jan Mayen	SJ
Swaziland	SZ
Sweden	SE
Switzerland	CH
Syrian Arab Republic	SY
Taiwan, Province of China	TW
Tajikistan	TJ
Tanzania, United Republic of	TZ
Thailand	TH
Timor-Leste	TL
Togo	TG
Tokelau	TK
Tonga	TO
Trinidad and Tobago	TT
Tunisia	TN
Turkey	TR
Turkmenistan	TM
Turks and Caicos Islands	TC
Tuvalu	TV
Uganda	UG
Ukraine	UA
United Arab Emirates	AE
United Kingdom	GB

United States	US
United States Minor Outlying Islands	UM
Uruguay	UY
Uzbekistan	UZ
Vanuatu	VU
Venezuela, Bolivarian Republic of	VE
Viet Nam	VN
Virgin Islands, British	VG
Virgin Islands, U.S.	VI
Wallis and Futuna	WF
Western Sahara	EH
Yemen	YE
Zambia	ZM
Zimbabwe	ZW

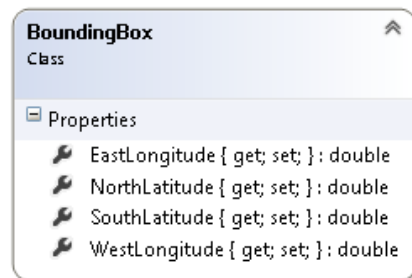
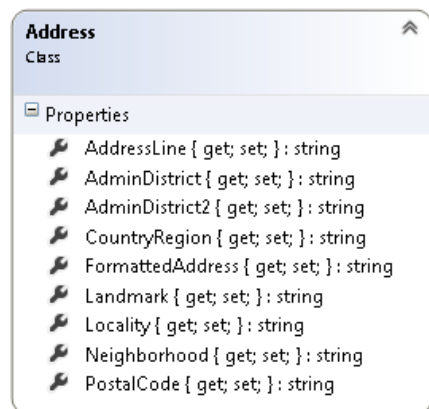
Appendix B: Bing Maps REST Services

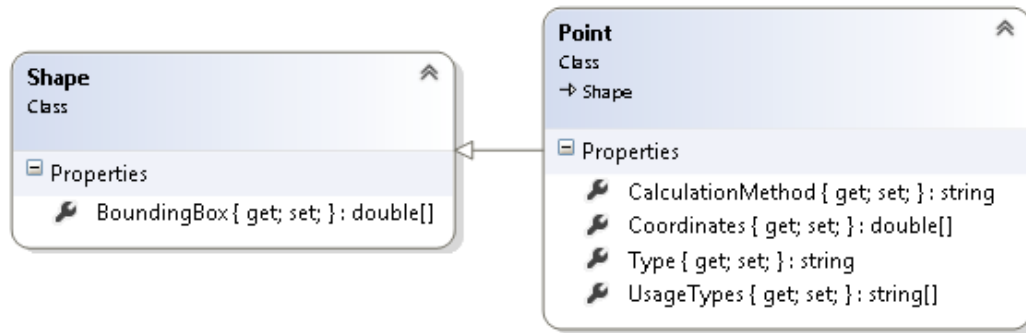
Class Diagrams

Base Response Class Diagram

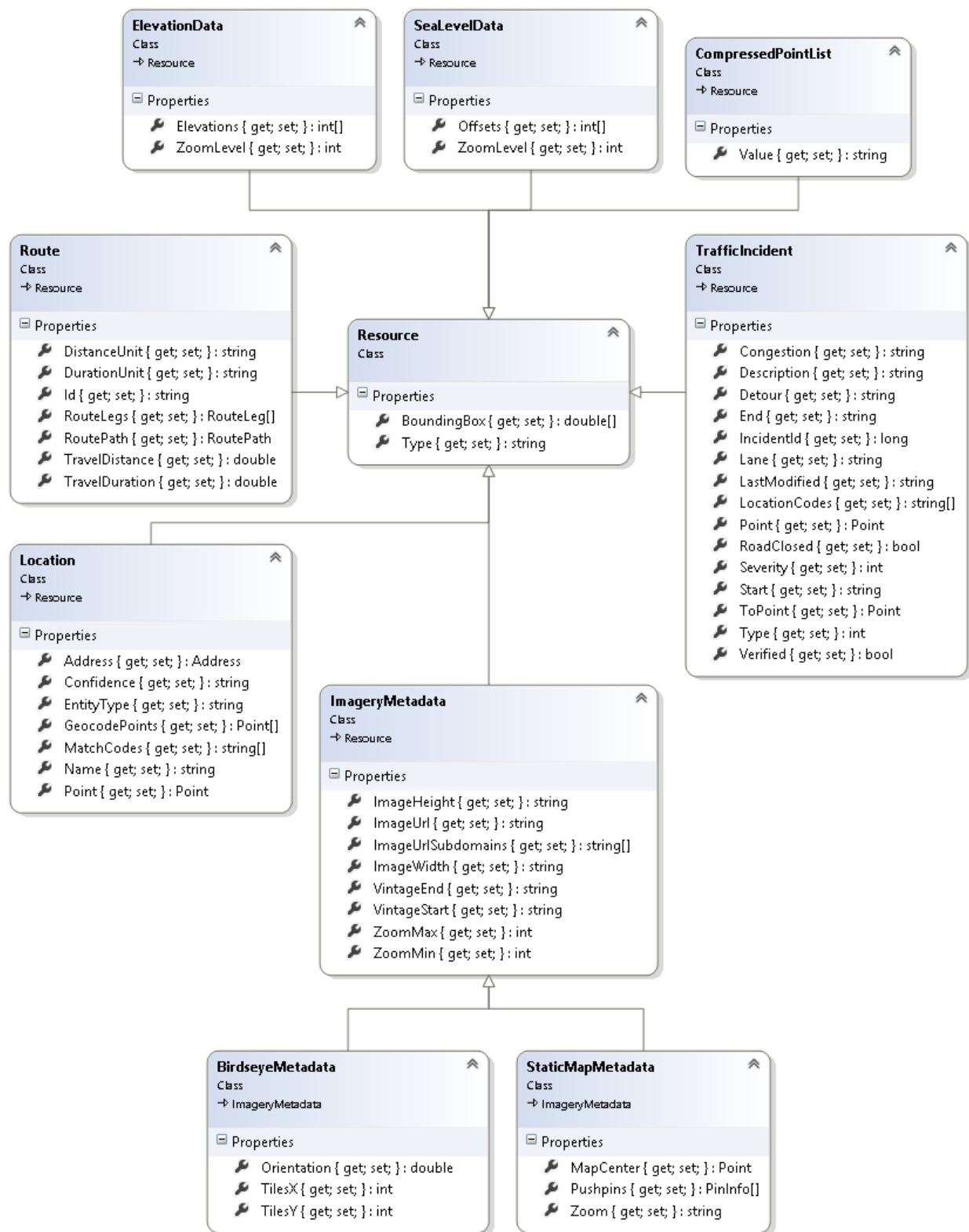


Common Base Classes

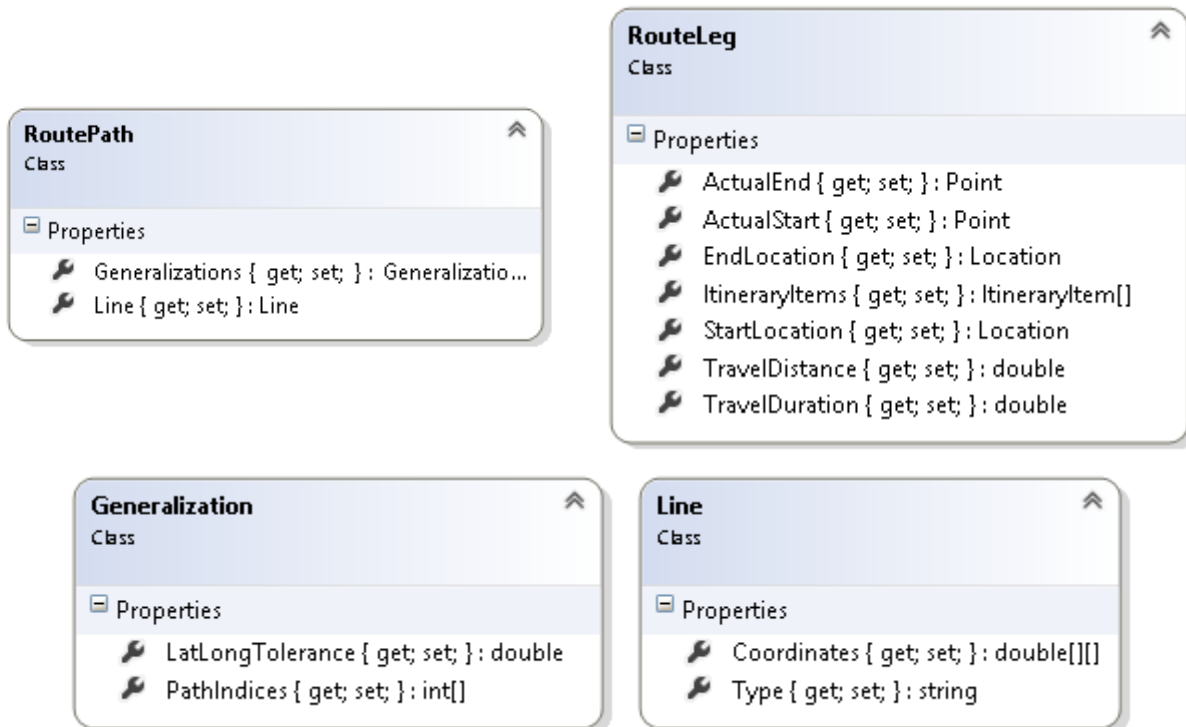


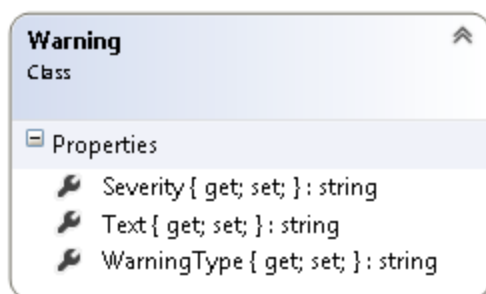
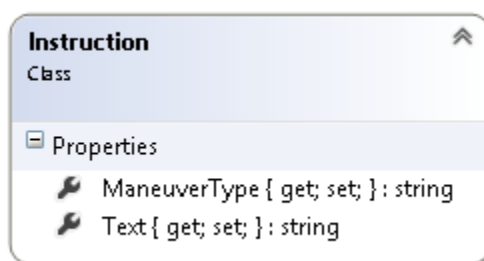
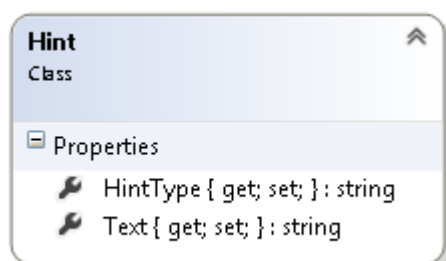
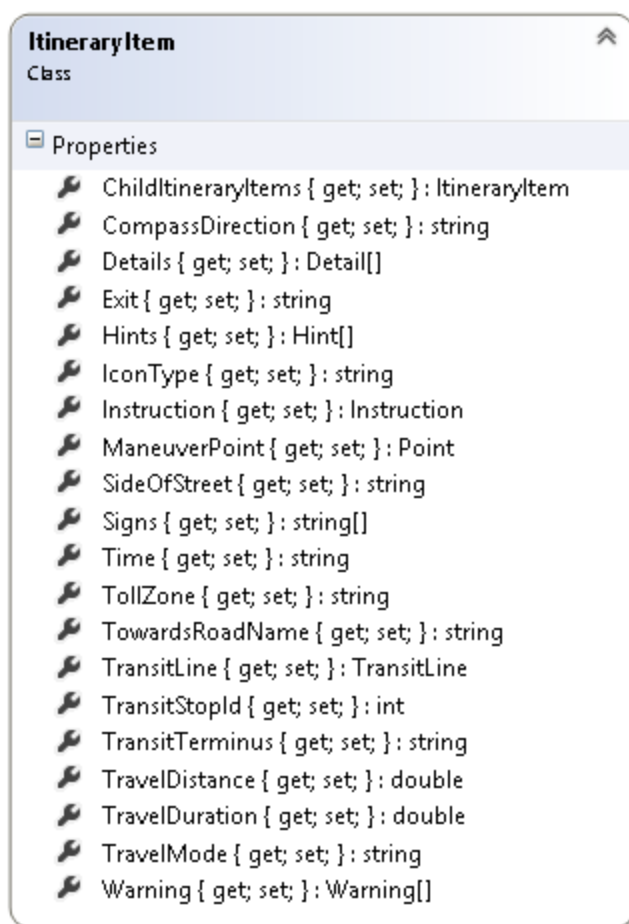


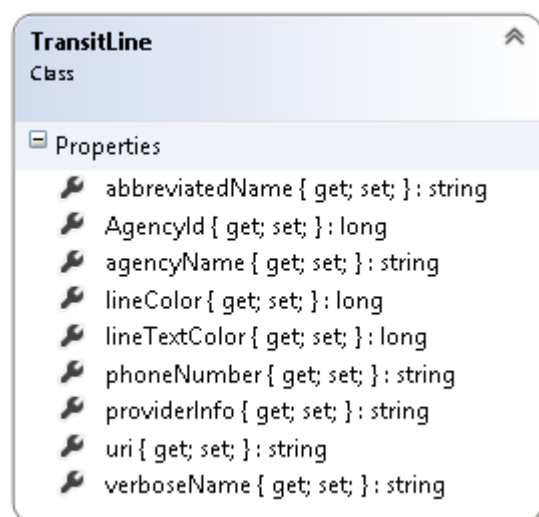
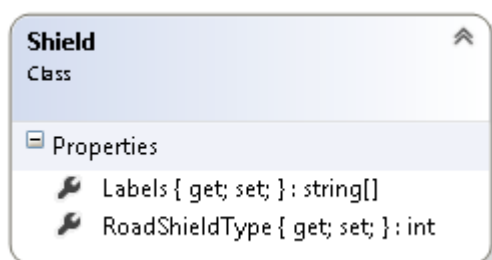
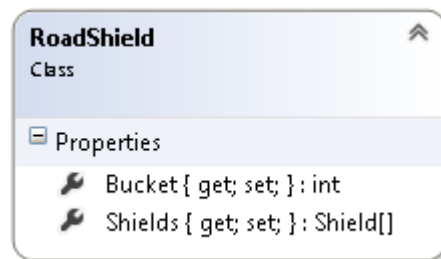
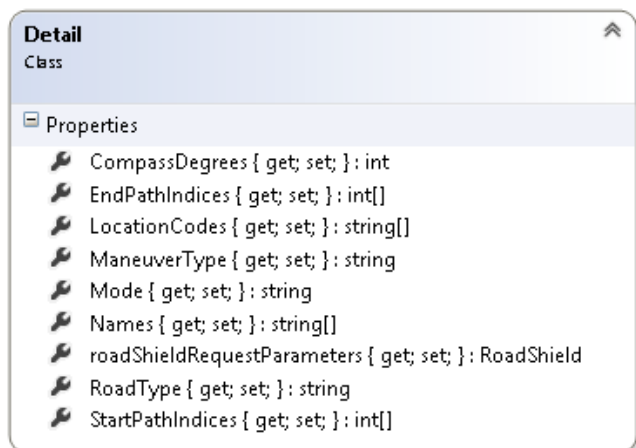
Resource Class Diagram



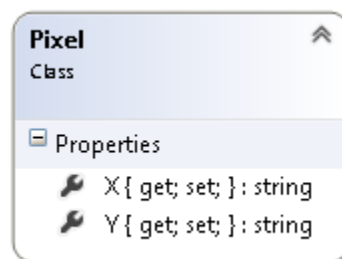
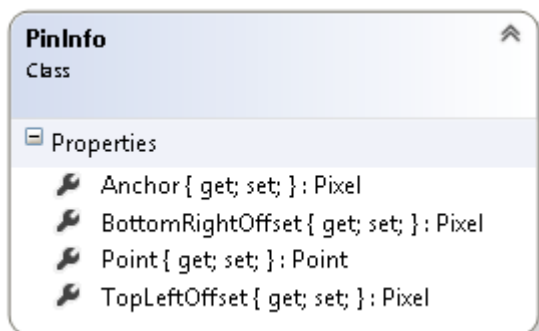
Routing Related Classes





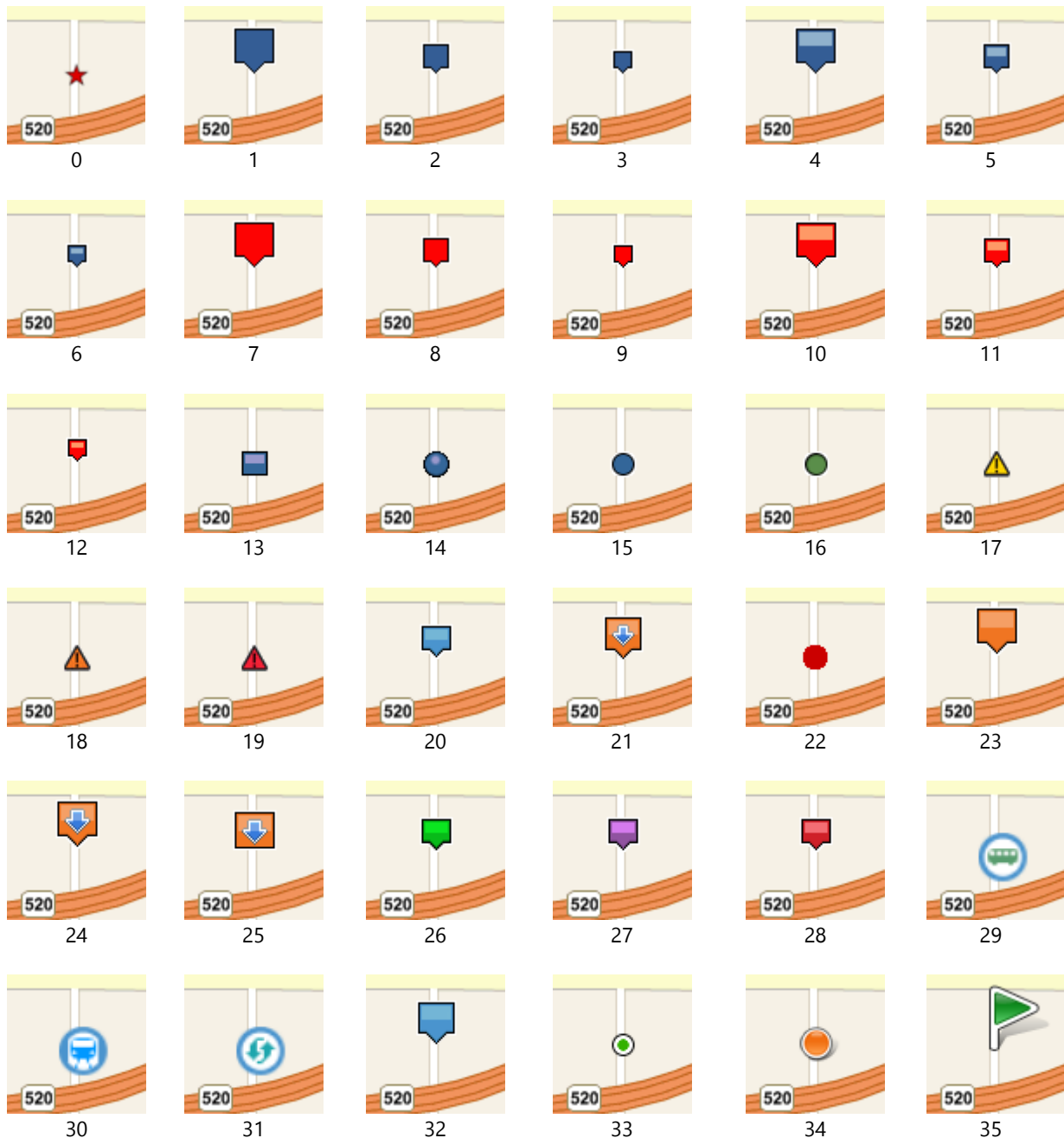


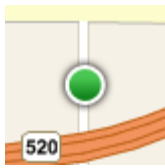
Imagery Related Classes



Appendix C: Bing Maps REST Service Pushpin Icon Styles

The following table shows the available pushpin icon styles and their associated id number that can be used with the Imagery API in the Bing Maps REST Services.





36



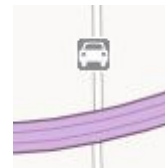
37



38



39



40



41



42



43



44



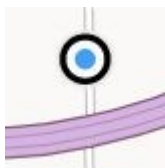
45



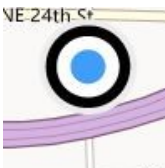
46



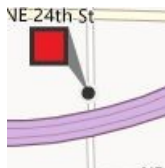
47



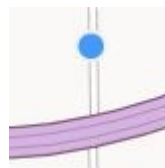
48



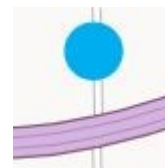
49



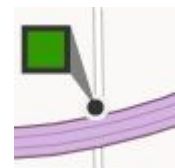
50



51



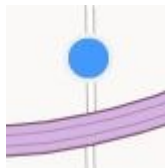
52



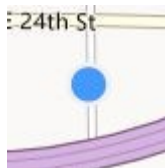
53



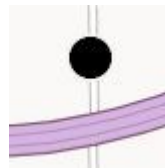
54



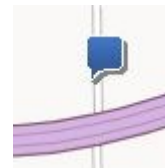
55



56



57



58



59



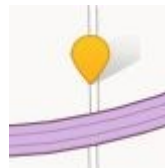
60



61



62



63



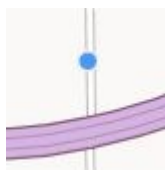
64



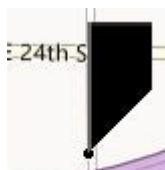
65



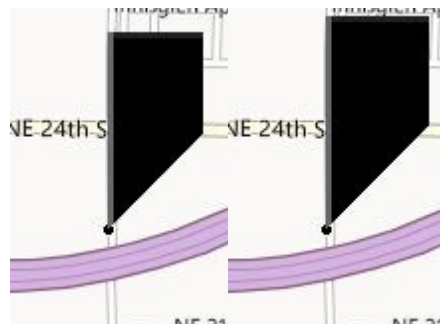
66



67



68



69



70



Appendix D: GeoData API Data Contracts

The following are the C# and Visual Basic data contract classes that can be used to deserialize the JSON responses from the GeoData API in the Bing Spatial Data Services.

C#

```
using System.Runtime.Serialization;

namespace GeoDataSchema
{
    [DataContract]
    public class Response
    {
        [DataMember(Name = "d", EmitDefaultValue = false)]
        public ResultSet ResultSet { get; set; }
    }

    [DataContract]
    public class ResultSet
    {
        [DataMember(Name = "__copyright", EmitDefaultValue = false)]
        public string Copyright { get; set; }

        [DataMember(Name = "results", EmitDefaultValue = false)]
        public Result[] Results { get; set; }
    }

    [DataContract]
    public class Result
    {
        [DataMember(Name = "EntityID", EmitDefaultValue = false)]
        public string EntityID { get; set; }

        [DataMember(Name = "EntityMetadata", EmitDefaultValue = false)]
        public Metadata EntityMetadata { get; set; }

        [DataMember(Name = "Name", EmitDefaultValue = false)]
        public Name Name { get; set; }

        [DataMember(Name = "Primitives", EmitDefaultValue = false)]
        public Primitive[] Primitives { get; set; }

        [DataMember(Name = "Copyright", EmitDefaultValue = false)]
        public Copyright Copyright { get; set; }
    }

    [DataContract]
    public class Metadata
    {
        [DataMember(Name = "AreaSqKm", EmitDefaultValue = false)]
        public string AreaSqKm { get; set; }
    }
}
```

```

    [DataMember(Name = "BestMapViewBox", EmitDefaultValue = false)]
    public string BestMapViewBox { get; set; }

    [DataMember(Name = "OfficialCulture", EmitDefaultValue = false)]
    public string OfficialCulture { get; set; }

    [DataMember(Name = "RegionalCulture", EmitDefaultValue = false)]
    public string RegionalCulture { get; set; }

    [DataMember(Name = "PopulationClass", EmitDefaultValue = false)]
    public string PopulationClass { get; set; }
}

[DataContract]
public class Name
{
    [DataMember(Name = "EntityName", EmitDefaultValue = false)]
    public string EntityName { get; set; }

    [DataMember(Name = "Culture", EmitDefaultValue = false)]
    public string Culture { get; set; }

    [DataMember(Name = "SourceID", EmitDefaultValue = false)]
    public string SourceID { get; set; }
}

[DataContract]
public class Primitive
{
    [DataMember(Name = "PrimitiveID", EmitDefaultValue = false)]
    public string PrimitiveID { get; set; }

    [DataMember(Name = "Shape", EmitDefaultValue = false)]
    public string Shape { get; set; }

    [DataMember(Name = "NumPoints", EmitDefaultValue = false)]
    public string NumPoints { get; set; }

    [DataMember(Name = "SourceID", EmitDefaultValue = false)]
    public string SourceID { get; set; }
}

[DataContract]
public class Copyright
{
    [DataMember(Name = "CopyrightURL", EmitDefaultValue = false)]
    public string CopyrightURL { get; set; }

    [DataMember(Name = "Sources", EmitDefaultValue = false)]
    public CopyrightSource[] Sources { get; set; }
}

[DataContract]
public class CopyrightSource
{
    [DataMember(Name = "SourceID", EmitDefaultValue = false)]
    public string SourceID { get; set; }

    [DataMember(Name = "SourceName", EmitDefaultValue = false)]
    public string SourceName { get; set; }
}

```

```

        [DataMember(Name = "Copyright", EmitDefaultValue = false)]
        public string Copyright { get; set; }
    }
}

```

Visual Basic

Imports System.Runtime.Serialization

Namespace GeoDataSchema

```

    <DataContract> _
    Public Class Response
        <DataMember(Name:="d", EmitDefaultValue:=False)> _
        Public Property ResultSet As ResultSet
    End Class

    <DataContract> _
    Public Class ResultSet
        <DataMember(Name:="__copyright", EmitDefaultValue:=False)> _
        Public Property Copyright As String

        <DataMember(Name:="results", EmitDefaultValue:=False)> _
        Public Property Results As Result()
    End Class

```

```

    <DataContract> _
    Public Class Result
        <DataMember(Name:="EntityID", EmitDefaultValue:=False)> _
        Public Property EntityID As String

        <DataMember(Name:="EntityMetadata", EmitDefaultValue:=False)> _
        Public Property EntityMetadata As Metadata

        <DataMember(Name:="Name", EmitDefaultValue:=False)> _
        Public Property Name As Name

        <DataMember(Name:="Primitives", EmitDefaultValue:=False)> _
        Public Property Primitives As Primitive()

        <DataMember(Name:="Copyright", EmitDefaultValue:=False)> _
        Public Property Copyright As Copyright
    End Class

```

```

    <DataContract> _
    Public Class Metadata
        <DataMember(Name:="AreaSqKm", EmitDefaultValue:=False)> _
        Public Property AreaSqKm As String

        <DataMember(Name:="BestMapViewBox", EmitDefaultValue:=False)> _
        Public Property BestMapViewBox As String

        <DataMember(Name:="OfficialCulture", EmitDefaultValue:=False)> _
        Public Property OfficialCulture As String

        <DataMember(Name:="RegionalCulture", EmitDefaultValue:=False)> _
        Public Property RegionalCulture As String
    End Class

```

```

    <DataMember(Name:="PopulationClass", EmitDefaultValue:=False)> _
    Public Property PopulationClass As String
End Class

<DataContract> _
Public Class Name
    <DataMember(Name:="EntityName", EmitDefaultValue:=False)> _
    Public Property EntityName As String

    <DataMember(Name:="Culture", EmitDefaultValue:=False)> _
    Public Property Culture As String

    <DataMember(Name:="SourceID", EmitDefaultValue:=False)> _
    Public Property SourceID As String
End Class

<DataContract> _
Public Class Primitive
    <DataMember(Name:="PrimitiveID", EmitDefaultValue:=False)> _
    Public Property PrimitiveID As String

    <DataMember(Name:="Shape", EmitDefaultValue:=False)> _
    Public Property Shape As String

    <DataMember(Name:="NumPoints", EmitDefaultValue:=False)> _
    Public Property NumPoints As String

    <DataMember(Name:="SourceID", EmitDefaultValue:=False)> _
    Public Property SourceID As String
End Class

<DataContract> _
Public Class Copyright
    <DataMember(Name:="CopyrightURL", EmitDefaultValue:=False)> _
    Public Property CopyrightURL As String

    <DataMember(Name:="Sources", EmitDefaultValue:=False)> _
    Public Property Sources As CopyrightSource()
End Class

<DataContract> _
Public Class CopyrightSource
    <DataMember(Name:="SourceID", EmitDefaultValue:=False)> _
    Public Property SourceID As String

    <DataMember(Name:="SourceName", EmitDefaultValue:=False)> _
    Public Property SourceName As String

    <DataMember(Name:="Copyright", EmitDefaultValue:=False)> _
    Public Property Copyright As String
End Class
End Namespace

```

Glossary

Aerial Imagery: This type of map overlays imagery that has been captured by planes and satellites pointing straight down. Bing Maps has some of the highest resolution imagery of this type available from an online map provider. In addition to that they also have one of the most up to date and complete data sets of this type of imagery as well.

Antemeridian: Also known as the 180th Meridian is the point where -180 degrees and 180 degrees of longitude meet. Which is on the completely opposite the prime meridian on the globe.

Augmented Reality: Is when an object that is not present in the real world, but appears to be because of a view showing a modified version of reality. This is often done by overlaying virtual items on top of a live video stream which is in line with the user's field of view. This gives the user a window into an augmented reality.

Bing Maps Key: A Bing Maps Key is a unique string that is used to authenticate a user's Bing Maps application or service request. This is the primary method used for tracking usage of the Bing Maps API's.

Bing Maps Session: A Bing Maps session occurs when one of the map controls is loaded. Any transactions occurred against the Bing Maps services, while within a session is non-billable. Note this requires the application to properly use Bing Maps Keys.

Bing Maps Transaction: A Bing Maps transaction occurs any time a service request is made. For example, some of the more common services used that incur transactions are: Bing Maps Geocoding, Routing, and Imagery service, Bing Spatial Data Service Query.

Birdseye Imagery: Birdseye imagery is captured by low-flying aircraft at a 45 degrees angle. This type of imagery is typically captured from 4 different directions. This gives the user the ability to rotate the imagery around a location and offers better depth perception for buildings and geography. These images are typically much more detailed than the aerial views taken from directly above.

Bounding Box: A set of coordinates used to represent a rectangular area on the map.

Byte-code caching: Byte-code caching is a technique that creates byte-code for each JavaScript file once, rather than recreating the byte-code each time it launches the app. This is very similar to compiling code ahead of time.

Coordinate: Consists of the latitude and longitude values used to represent a location on a map.

CSS: An acronym for Cascading Style Sheet which is used to store style information for web pages.

Geocode: The process of converting an address into a coordinate that can be displayed on a map.

Geodesic Path: The shortest path between two points on a curved surface. When rendered on Bing Maps this path appears as a curved line due to the Mercator projection.

Geofence: A defined geographical region which can be used to trigger events when a device enters or exists the region.

GIS: An acronym for "Geographic Information System". A common term used to describe the mapping industry.

Heat map: See Thematic Map.

Latitude: The angular distance measured in degrees from equator in a north or south direction.

Linestring: A shape used to represent a line. This is sometimes also called a polyline. This term is often used when working with the spatial functionalities in SQL Server.

Longitude: The angular distance measured in degrees from the prime meridian in an east or west direction.

Lumens: A lumen is a standard unit of measurement used to represent the total amount of visible light emitted by a source.

Lux: A lux is a standard unit of measurement that represents the total amount of visible light in a square meter. This is equivalent to one lumen per square meter.

OGC: An acronym for “Open Geospatial Consortium”. The OGC is an international volunteer body that sets defines open standards for geospatial content and services, GIS data processing and data sharing. Systems that can understand and/or output data in these formats are said to be “OGC Compliant”.

Ordnance Survey: The national mapping agency of Great Britain that was established in 1791. It focuses on collecting the most detailed spatial data in Great Britain and producing maps.

Polygon: A solid shape. Often used to mark out boundaries.

Polyline: A shape used to represent a line. This is sometimes also referred to as a LineString.

Prime Meridian: A line of longitude that represents 0 degrees longitude. Generally the longitude values increase when traveling in a westerly direction until 180 degrees is reached, and decreases when traveling in easterly directions to -180 degrees.

Pushpin: Marker or icon used to pin point a location on a map.

Quaternion: A quaternion represents two things, an axis on which a rotation occurs and the rotation around that axis. It has an x, y, and z component, which represents the axis on which the rotation occurs. It also has a w component, which represents the amount of rotation that occurs around this axis.

Raster data: A cell-based type of data such as aerial or satellite imagery.

REST Service: The acronym REST stands for Representational State Transfer. A REST service is a URL based web service that relies on basic web technology to communicate, the most common methods being HTTP GET and POST requests. These types of services tend to be much quicker and smaller than traditional SOAP based services.

Reverse Geocode: The process of taking a coordinate and determining the address in which it represents on a map.

Road Maps: Road maps are one of the more basic types of maps offered in Bing Maps. This map style was designed to be more of a canvas for overlaying data such that the data will stand out better. In addition to that it has also been designed for those with color blindness in mind.

Rotation Matrix: A rotation matrix is a matrix that is used to perform a rotation in the Cartesian coordinate system.

Thematic maps: A thematic map is a simple map made to reflect a theme about a geographic area. A common scenario for this type of map is to color the administrative regions such as countries based on some metric of data.

Vector data: Coordinate based data that is represented as points, polylines, or polygons.

Venue Maps: Venue Maps are often maps of indoor structures such as malls and airports, however venue maps are not just limited to this. Other types of venue maps include the layout of shopping districts, stadiums, race courses, and universities. Bing Maps has thousands of venue maps available around the world.

Virtual Reality: Often confused with augmented reality. Virtual Reality consists of a virtual simulated world.

Waypoint: A waypoint is a specified geographical location defined by longitude and latitude that is used for navigational purposes.

WGS84: A set of constants used to relate spatial coordinates to locations on the surface of the map. The WGS84 datum is the standard one used by most online mapping providers and GPS devices.

WinJS: This is a namespace that provides a special Windows Library for JavaScript functionality in Windows Store apps. It is often useful to include “WinJS” rather than “Windows Store” when doing online searches related to the development of JavaScript Windows Store apps.

WinRT: An acronym for Windows Runtime. It is often useful to include “WinRT” rather than “Windows Store” when doing online searches related to the development of native Windows Store apps.

Location Intelligence for Windows Store Apps

Location Intelligence has been one of the fastest growing industries in recent years and continues to grow at an exponential rate. Seventy to eighty percent of all business data has some sort of geospatial context. Many companies want to make use of this data however most of them do not know where to start. Many of these same companies are planning to create applications targeting Windows 8.

In this book we will dive into the world of location intelligence and the different options for creating location aware applications in Windows 8.1. The first half of the book will focus on the inner workings of Window Store Apps and the various location related tools available such as sensors and the Bing Maps SDK. The second half of the book focuses on creating several useful location intelligent apps. All code samples are provided in JavaScript, C# and Visual Basic.

What You'll Learn

- What sensors are available to Windows Store apps and how to use them.
- How to use the Bing Maps SDK for Windows Store apps.
- What other Bing Maps services can be used with Windows Store apps.
- How to work with spatial data and easily import or export it from Bing Maps.
- How to create a simple augmented reality app.
- How to draw data on a map using mouse and touch events.
- The basics of cross platform development using tools such as JavaScript, Conditional Compilation Symbols, and Portable Class libraries.

ISBN 978-1-291-76109-2



9 781291 761092