

Contents

[Azure Maps documentation](#)

[Overview](#)

[What is Azure Maps?](#)

[Quickstarts](#)

[Create a web app](#)

[Create an Android app](#)

[Tutorials](#)

[Web SDK Tutorials](#)

[Search for point of interest](#)

[Route to a destination](#)

[Multiple routes by mode of travel](#)

[Create a store locator](#)

[Android SDK Tutorials](#)

[Load GeoJSON data into map](#)

[REST API Tutorials](#)

[Create indoor maps](#)

[Set up a geofence](#)

[Spatial analytics](#)

[EV routing using Azure Notebooks \(Python\)](#)

[Analyze weather data using Azure Notebooks \(Python\)](#)

[Migrate from Bing Maps](#)

[Migrate from Bing Maps](#)

[Migrate a web app](#)

[Migrate a web service](#)

[Migrate from Google Maps](#)

[Migrate from Google Maps](#)

[Migrate a web app](#)

[Migrate an Android app](#)

[Migrate a web service](#)

Concepts

[Authentication with Azure Maps](#)

[Azure Maps Event Grid integration](#)

[Choose the right pricing tier](#)

[Creator for Indoor maps](#)

[Creator service geographic scope](#)

[Indoor map concepts](#)

[Drawing conversion errors and warnings](#)

[Encryption of data at rest](#)

Coverage

[Coverage](#)

[Geocoding coverage](#)

[Traffic coverage](#)

[Render coverage](#)

[Routing coverage](#)

[Mobility coverage](#)

[Weather coverage](#)

[Localization support](#)

[Supported map styles](#)

[Zoom levels and tile grid](#)

[Mobility services](#)

[Weather services concepts](#)

[Weather services FAQ](#)

How-to guides

[Manage Maps accounts](#)

[Create account with ARM template](#)

[Manage accounts](#)

[Manage authentication](#)

[How to secure a daemon application](#)

[How to secure interactive sign-in single page application](#)

[How to secure web application](#)

[How to secure non-interactive sign-in single page application](#)

[How to secure input constrained application](#)

[Manage pricing tier](#)

[Manage Creator](#)

[View usage metrics](#)

[Get map data from REST APIs](#)

[Search for an address](#)

[Best Practices using Search Service](#)

[Best Practices using Route Service](#)

[Render custom data on static map](#)

[Request public transit routes](#)

[Request real-time public transit data](#)

[Request real-time and forecasted weather data](#)

[Request elevation data](#)

[Develop with the Web SDK](#)

[Map control](#)

[Use the Azure Maps map control](#)

[Create a map](#)

[Change the style of the map](#)

[Add controls to the map](#)

[Create a data source](#)

[Add a symbol layer](#)

[Add a bubble layer](#)

[Add a popup](#)

[Add HTML markers](#)

[Add a line layer](#)

[Add a polygon layer](#)

[Add a polygon extrusion layer](#)

[Add a heat map layer](#)

[Add an image layer](#)

[Add tile layers](#)

[Show traffic](#)

[Cluster point data](#)

- [Use data-driven style expressions](#)
- [How to use image templates](#)
- [Reacting to events](#)
- [Make your app accessible](#)
- [Drawing tools](#)
 - [Use the drawing tools module](#)
 - [Add a drawing toolbar](#)
 - [Get shape data](#)
 - [React to drawing events](#)
 - [Interaction types and keyboard shortcuts](#)
- [Services module](#)
 - [Use the services module](#)
 - [Show search results](#)
 - [Get information from a coordinate](#)
 - [Show directions from A to B](#)
- [Indoor Maps module](#)
 - [Indoor Maps module](#)
 - [Dynamic styling for indoor maps](#)
 - [Drawing error visualizer](#)
- [Spatial IO module](#)
 - [Use the spatial IO module](#)
 - [Add a simple data layer](#)
 - [Read and write spatial data](#)
 - [Add an OGC map layer](#)
 - [Connect to a WFS service](#)
 - [Leverage core operations](#)
 - [Supported data format details](#)
- [Web SDK supported browsers](#)
- [Best practices](#)
- [Develop with the Android SDK](#)
 - [Getting started with Android map control](#)
 - [Change the style of the map](#)

- [Add controls to the map](#)
- [Create a data source](#)
- [Add a symbol layer](#)
- [Add a bubble layer](#)
- [Display feature information](#)
- [Add a line layer](#)
- [Add a polygon layer](#)
- [Add a polygon extrusion layer](#)
- [Add a heat map layer](#)
- [Add an image layer](#)
- [Add a tile layer](#)
- [Show traffic data](#)
- [Use data-driven expressions](#)
- [Reacting to events](#)

[Use Azure Maps in Power BI](#)

- [Getting started](#)
- [Understanding layers](#)
- [Add a bubble layer](#)
- [Add a bar chart layer](#)
- [Add a reference layer](#)
- [Add a tile layer](#)
- [Show real-time traffic](#)
- [Manage access](#)

[Provide data feedback](#)

[Reference](#)

- [API and parameter extensions](#)
- [Drawing package requirements](#)
- [Drawing package guide](#)
- [Dynamic maps StylesObject](#)
- [Vehicle consumption model](#)
- [Extended GeoJSON format](#)
- [Geofence GEOJSON format](#)

[Supported search categories](#)

[Facility Ontology](#)

[REST](#)

[Data service](#)

[Elevation service](#)

[Geolocation service](#)

[Mobility services](#)

[Render service](#)

[Render service V2](#)

[Route service](#)

[Search service](#)

[Spatial service](#)

[Timezone service](#)

[Traffic service](#)

[Weather services](#)

[Maps Creator service](#)

[Alias service](#)

[Conversion service](#)

[Dataset service](#)

[Feature State service](#)

[Tileset service](#)

[WFS service](#)

[Maps account management](#)

[JavaScript](#)

[Map control](#)

[Drawing tools](#)

[Service module](#)

[Spatial IO module](#)

[Indoor module](#)

[Resource Manager template](#)

[Azure CLI](#)

[Resources](#)

[Support and help options](#)

[Azure IoT services](#)

[IoT Hub](#)

[IoT Hub Device Provisioning Service](#)

[IoT Central](#)

[IoT Edge](#)

[IoT solution accelerators](#)

[IoT Plug and Play](#)

[Azure Maps](#)

[Time Series Insights](#)

[Azure IoT SDKs](#)

[IoT Service SDKs](#)

[IoT Device SDKs](#)

[Azure Maps terms of use](#)

[Azure Maps pricing](#)

[Pricing calculator](#)

[Azure Maps videos](#)

[Open source projects](#)

[Azure Maps case studies](#)

[Azure Maps glossary](#)

[Service updates](#)

[Azure IoT Developer Center](#)

[Azure Roadmap](#)

[Technical case studies](#)

[Azure Maps on IoT School](#)

What is Azure Maps?

6/1/2021 • 8 minutes to read • [Edit Online](#)

Azure Maps is a collection of geospatial services and SDKs that use fresh mapping data to provide geographic context to web and mobile applications. Azure Maps provides:

- REST APIs to render vector and raster maps in multiple styles and satellite imagery.
- Creator services to create and render maps based on private indoor map data.
- Search services to locate addresses, places, and points of interest around the world.
- Various routing options; such as point-to-point, multipoint, multipoint optimization, isochrone, electric vehicle, commercial vehicle, traffic influenced, and matrix routing.
- Traffic flow view and incidents view, for applications that require real-time traffic information.
- Mobility services (Preview) to request public transit information, plan routes by blending different travel modes and real-time arrivals.
- Time zone and Geolocation (Preview) services.
- Elevation services with Digital Elevation Model
- Geofencing service and mapping data storage, with location information hosted in Azure.
- Location intelligence through geospatial analytics.

Additionally, Azure Maps services are available through the Web SDK and the Android SDK. These tools help developers quickly develop and scale solutions that integrate location information into Azure solutions.

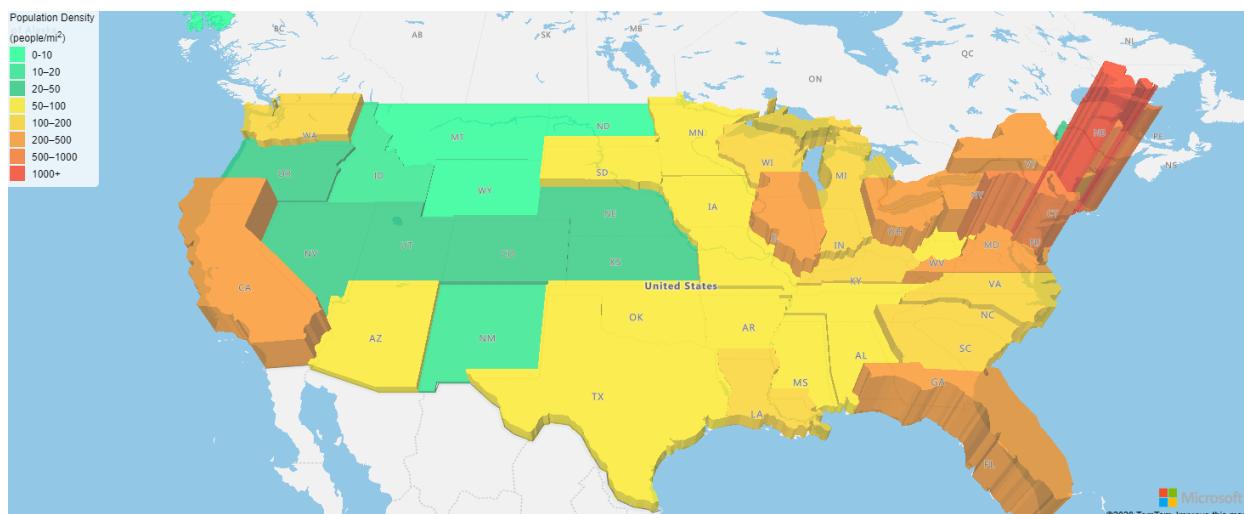
You can sign up for a free [Azure Maps account](#) and start developing.

The following video explains Azure Maps in depth:

Map controls

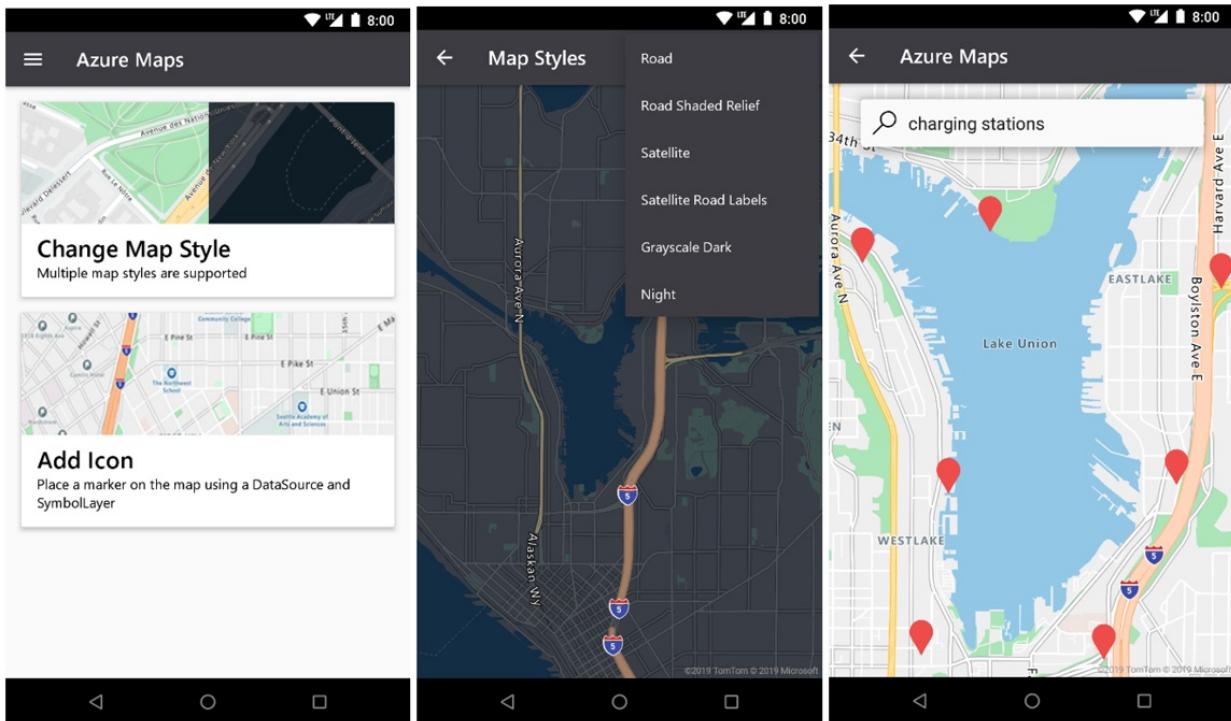
Web SDK

The Azure Maps Web SDK lets you customize interactive maps with your own content and imagery. You can use this interactive map for both your web or mobile applications. The map control makes use of WebGL, so you can render large data sets with high performance. You can develop with the SDK by using JavaScript or TypeScript.



Android SDK

Use the Azure Maps Android SDK to create mobile mapping applications.



Services in Azure Maps

Azure Maps consists of the following services that can provide geographic context to your Azure applications.

Data service

Data is imperative for maps. Use the Data service to upload and store geospatial data for use with spatial operations or image composition. Bringing customer data closer to the Azure Maps service will reduce latency, increase productivity, and create new scenarios in your applications. For details on this service, see the [Data service documentation](/rest/api/maps/data v2).

Geolocation service (Preview)

Use the Geolocation service to preview the retrieved two-letter country/region code for an IP address. This service can help you enhance user experience by providing customized application content based on geographic location.

For more details, read the [Geolocation service documentation](#).

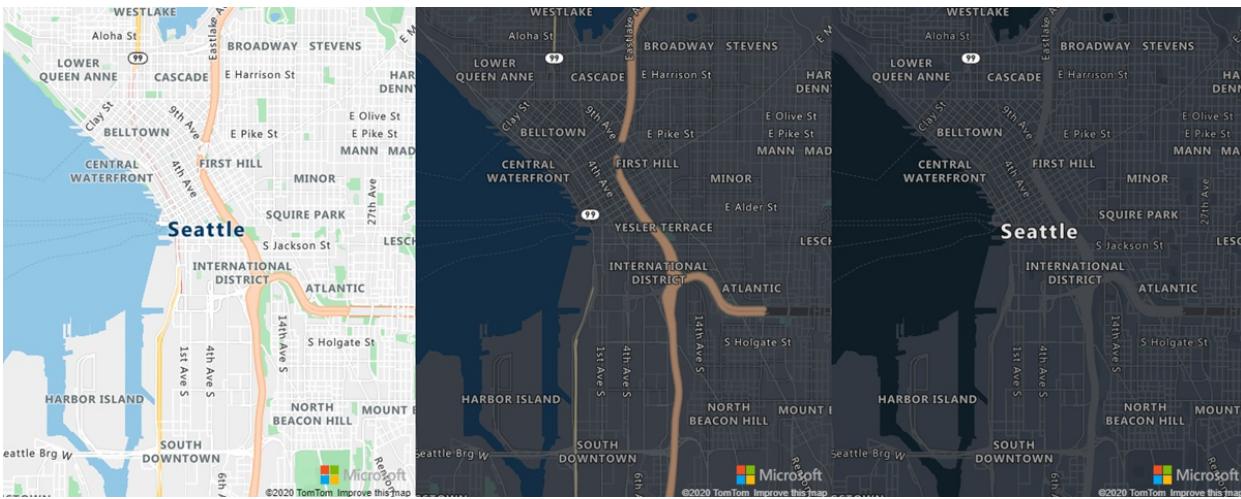
Mobility services (Preview)

The Azure Maps Mobility services improve the development time for applications with public transit features, such as transit routing and search for nearby public transit stops. Users can retrieve detailed information about transit stops, lines, and schedules. The Mobility service also allows users to retrieve stop and line geometries, alerts for stops, lines, and service areas, and real-time public transit arrivals and service alerts. Additionally, the Mobility services provide routing capabilities with multimodal trip planning options. Multimodal trip planning incorporates walking, bicycling, and public transit options, all into one trip. Users can also access detailed multimodal step-by-step itineraries.

To learn more about the service, see the [Mobility services documentation](#).

Render service

The [Render service V2 \(Preview\)](#) introduces a new version of the [Get Map Tile V2 API](#). The Get Map Tile V2 API now allows customers to request Azure Maps road tiles, weather tiles, or the map tiles created using Azure Maps Creator. It's recommended that you use the new Get Map Tile V2 API.

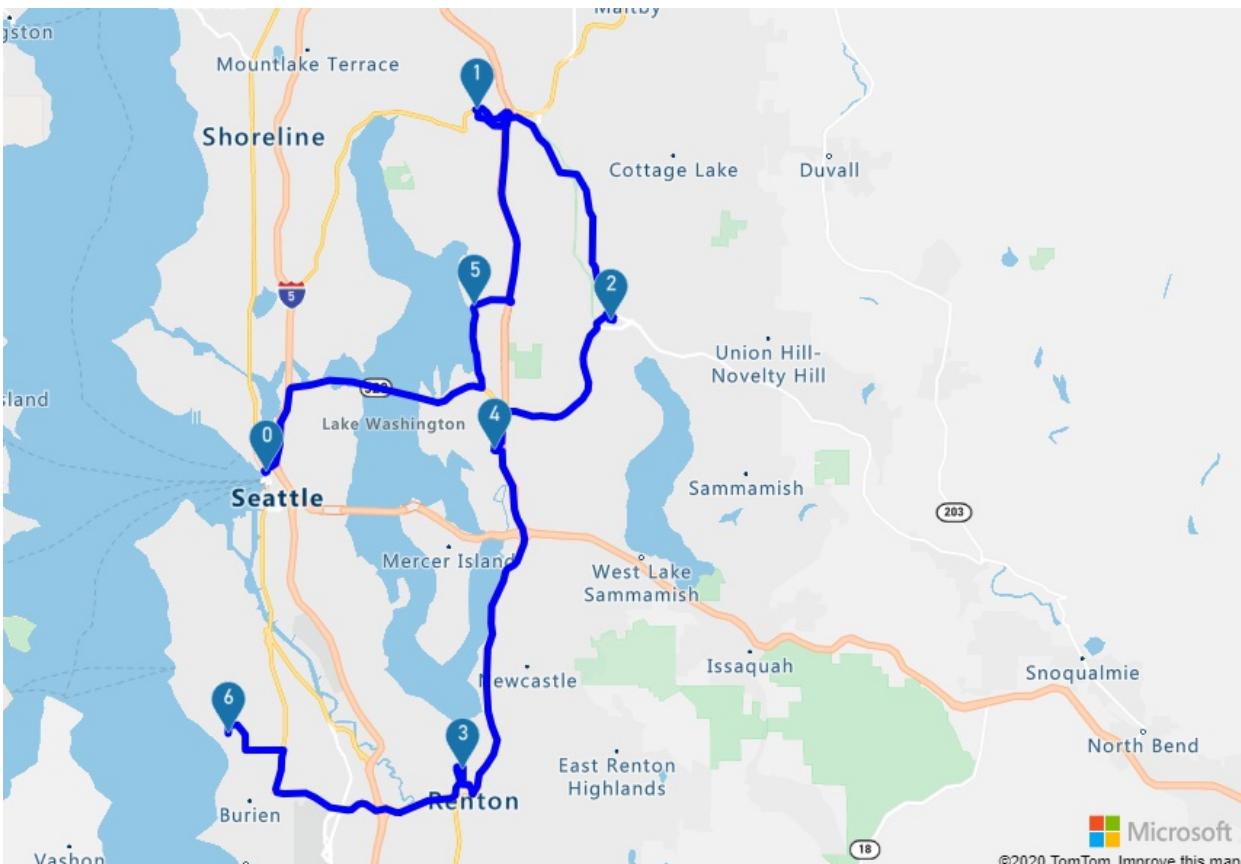


For more details, read the [Render service V2 documentation](#).

To learn more about the Render service V1 that is in GA (General Availability), see the [Render service V1 documentation](#).

Route service

The route services can be used to calculate the estimated arrival times (ETAs) for each requested route. Route APIs consider factors, such as real-time traffic information and historic traffic data, like the typical road speeds on the requested day of the week and time of day. The APIs return the shortest or fastest routes available to multiple destinations at a time in sequence or in optimized order, based on time or distance. The service allows developers to calculate directions across several travel modes, such as car, truck, bicycle, or walking, and electric vehicle. The service also considers inputs, such as departure time, weight restrictions, or hazardous material transport.



The Route service offers advanced set features, such as:

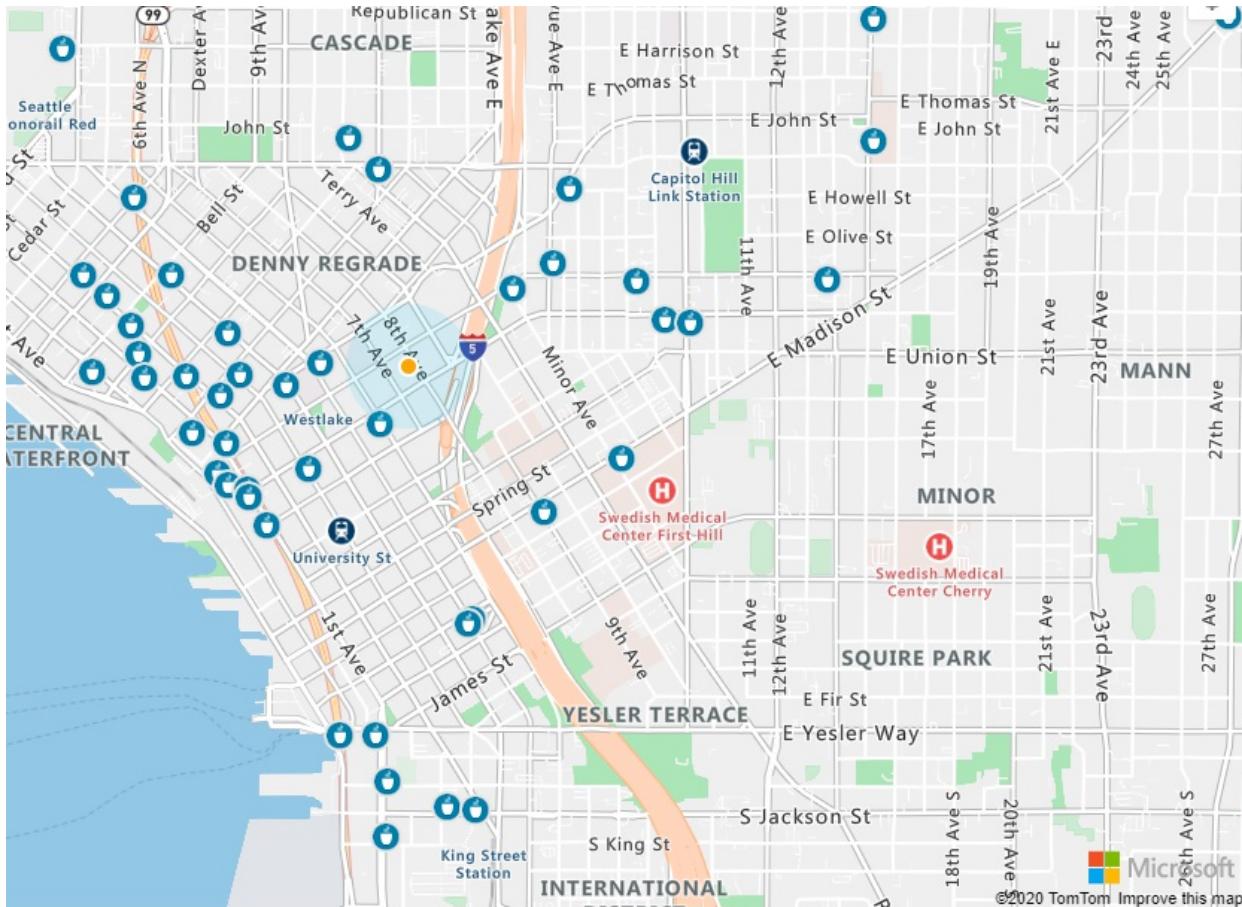
- Batch processing of multiple route requests.
- Matrices of travel time and distance between a set of origins and destinations.

- Finding routes or distances that users can travel based on time or fuel requirements.

For details on the routing capabilities, read the [Route service documentation](#).

Search service

The Search service helps developers search for addresses, places, business listings by name or category, and other geographic information. Also, services can [reverse geocode](#) addresses and cross streets based on latitudes and longitudes.



The Search service also provides advanced features such as:

- Search along a route.
- Search inside a wider area.
- Batch a group of search requests.
- Search electric vehicle charging stations and Point of Interest (POI) data by brand name.

For more details on search capabilities, read the [Search service documentation](#).

Spatial service

The Spatial service quickly analyzes location information to help inform customers of ongoing events happening in time and space. It enables near real-time analysis and predictive modeling of events.

The service enables customers to enhance their location intelligence with a library of common geospatial mathematical calculations. Common calculations include closest point, great circle distance, and buffers. To learn more about the service and the various features, read the [Spatial service documentation](#).

Timezone service

The Time zone service enables you to query current, historical, and future time zone information. You can use either latitude and longitude pairs or an [IANA ID](#) as an input. The Time zone service also allows for:

- Converting Microsoft Windows time-zone IDs to IANA time zones.

- Fetching a time-zone offset to UTC.
- Getting the current time in a chosen time zone.

A typical JSON response for a query to the Time zone service looks like the following sample:

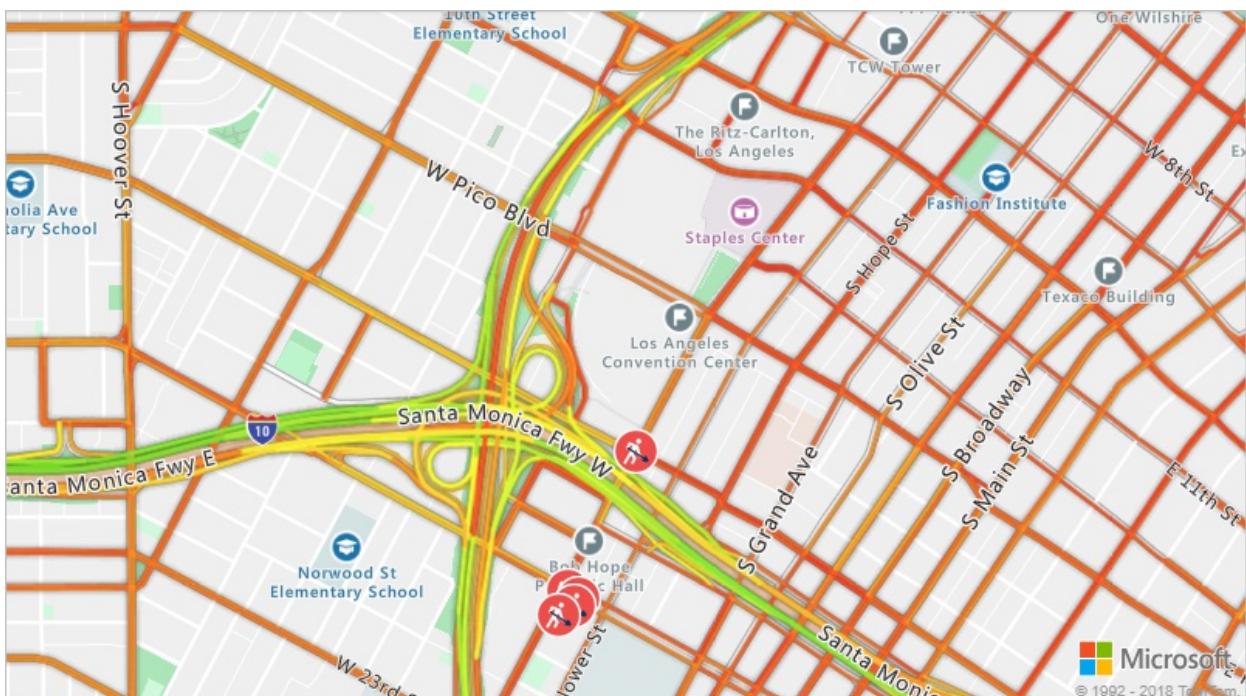
```
{
  "Version": "2020a",
  "ReferenceUtcTimestamp": "2020-07-31T19:15:14.4570053Z",
  "TimeZones": [
    {
      "Id": "America/Los_Angeles",
      "Names": {
        "ISO6391LanguageCode": "en",
        "Generic": "Pacific Time",
        "Standard": "Pacific Standard Time",
        "Daylight": "Pacific Daylight Time"
      },
      "ReferenceTime": {
        "Tag": "PDT",
        "StandardOffset": "-08:00:00",
        "DaylightSavings": "01:00:00",
        "WallTime": "2020-07-31T12:15:14.4570053-07:00",
        "PosixTzValidYear": 2020,
        "PosixTz": "PST+8PDT,M3.2.0,M11.1.0"
      }
    }
  ]
}
```

For details on this service, read the [Time zone service documentation](#).

Traffic service

The Traffic service is a suite of web services that developers can use for web or mobile applications that require traffic information. The service provides two data types:

- Traffic flow: Real-time observed speeds and travel times for all key roads in the network.
- Traffic incidents: An up-to-date view of traffic jams and incidents around the road network.



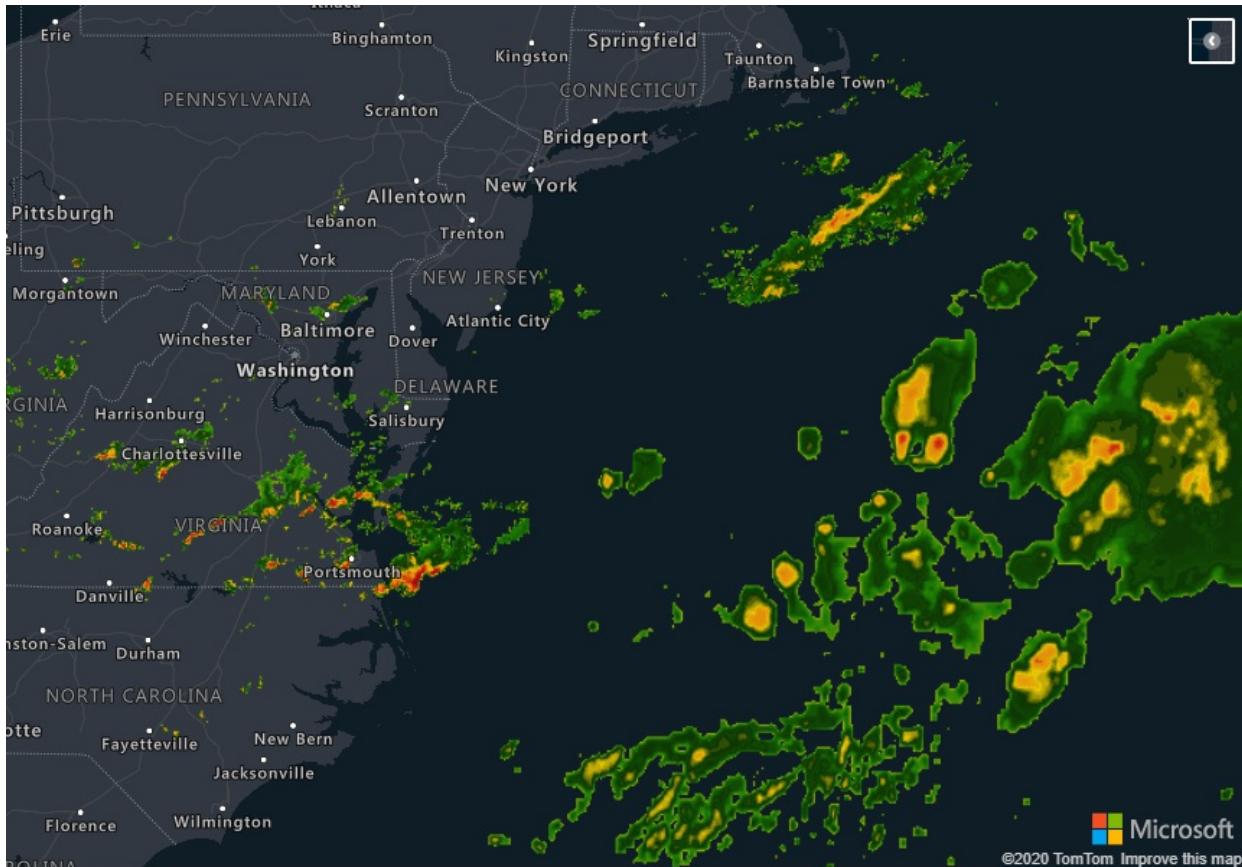
For more information, see the [Traffic service documentation](#).

Weather services

Weather services offer APIs that developers can use to retrieve weather information for a particular location. The information contains details such as observation date and time, brief description of the weather conditions, weather icon, precipitation indicator flags, temperature, and wind speed information. Additional details such as RealFeel™ Temperature and UV index are also returned.

Developers can use the [Get Weather along route API](#) to retrieve weather information along a particular route. Also, the service supports the generation of weather notifications for waypoints that are affected by weather hazards, such as flooding or heavy rain.

The [Get Map Tile V2 API](#) allows you to request past, current, and future radar and satellite tiles.



Maps Creator service

Maps Creator service is a suite of web services that developers can use to create applications with map features based on indoor map data.

Maps Creator provides three core services:

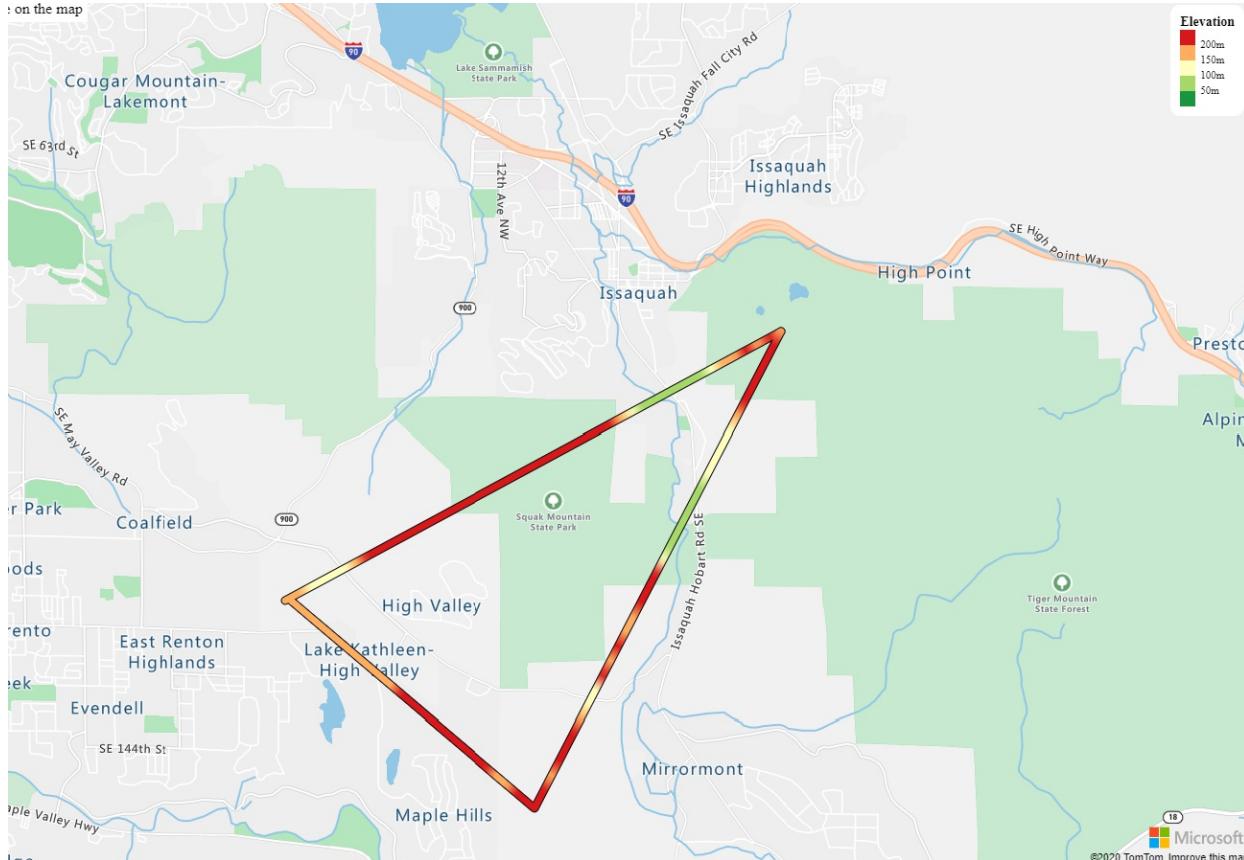
- [Dataset service](#). Use the Dataset service to create a dataset from a converted Drawing package data. For information about Drawing package requirements, see [Drawing package requirements](#).
- [Conversion service](#). Use the Conversion service to convert a DWG design file into Drawing package data for indoor maps.
- [Tileset service](#). Use the Tileset service to create a vector-based representation of a dataset. Applications can use a tileset to present a visual tile-based view of the dataset.
- [Feature State service](#). Use the Feature State service to support dynamic map styling. Dynamic map styling allows applications to reflect real-time events on spaces provided by IoT systems.
- [WFS service](#). Use the WFS service to query your indoor map data. The WFS service follows the [Open Geospatial Consortium API](#) standards for querying a single dataset.

Elevation service

The Azure Maps Elevation service is a web service that developers can use to retrieve elevation data from anywhere on the Earth's surface.

The Elevation service allows you to retrieve elevation data in two formats:

- **GeoTIFF raster format.** Use the [Render V2-Get Map Tile API](#) to retrieve elevation data in tile format.
- **GeoJSON format.** Use the [Elevation APIs](#) to request sampled elevation data along paths, within a defined bounding box, or at specific coordinates.



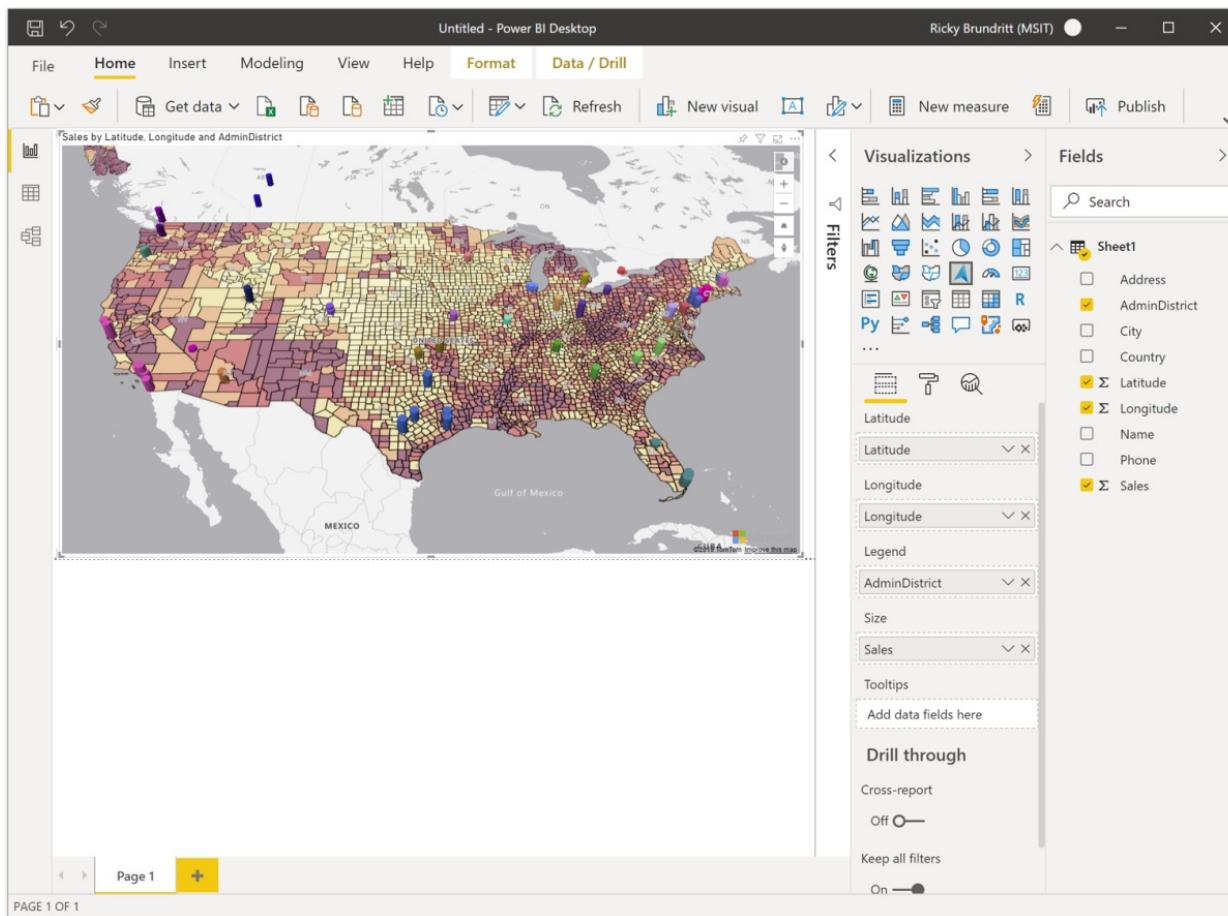
Programming model

Azure Maps is built for mobility and can help you develop cross-platform applications. It uses a programming model that's language agnostic and supports JSON output through [REST APIs](#).

Also, Azure Maps offers a convenient [JavaScript map control](#) with a simple programming model. The development is quick and easy for both web and mobile applications.

Power BI visual

The Azure Maps visual for Power BI provides a rich set of data visualizations for spatial data on top of a map. It's estimated that over 80% of business data has a location context. The Azure Maps visual offers a no-code solution for gaining insights into how this location context relates to and influences your business data.



For more information, see the [Getting started with the Azure Maps Power BI visual](#) documentation.

Usage

To access Azure Maps services, go to the [Azure portal](#) and create an Azure Maps account.

Azure Maps uses a key-based authentication scheme. When you create your account, two keys are generated. To authenticate for Azure Maps services, you can use either key.

Note - Azure Maps shares customer-provided address/location queries ("Queries") with third-party TomTom for mapping functionality purposes. Queries aren't linked to any customer or end user when shared with TomTom and can't be used to identify individuals. The Mobility and Weather services, which include integration with Moovit, and AccuWeather are currently in [PREVIEW](#).

Microsoft is currently in the process of adding TomTom, Moovit, and AccuWeather to the Online Services Subcontractor List.

Supported regions

Azure Maps services are currently available except in the following countries/regions:

- China
- South Korea

Verify that the location of your current IP address is in a supported country/region.

Next steps

Try a sample app that showcases Azure Maps:

[Quickstart: Create a web app](#)

Stay up to date on Azure Maps:

[Azure Maps blog](#)

Quickstart: Create an interactive search map with Azure Maps

4/30/2021 • 3 minutes to read • [Edit Online](#)

This article shows you how to use Azure Maps to create a map that gives users an interactive search experience. It walks you through these basic steps:

- Create your own Azure Maps account.
- Get your primary key to use in the demo web application.
- Download and open the demo map application.

This quickstart uses the Azure Maps Web SDK, however the Azure Maps services can be used with any map control. [Here](#) are some popular open-source map controls that the Azure Maps team has created plugin's for.

Prerequisites

- If you don't have an Azure subscription, create a [free account](#) before you begin.
- Sign in to the [Azure portal](#).

Create an Azure Maps account

Create a new Azure Maps account with the following steps:

1. In the upper left-hand corner of the [Azure portal](#), click **Create a resource**.
2. In the *Search the Marketplace* box, type **Azure Maps**.
3. From the *Results*, select **Azure Maps**. Click **Create** button that appears below the map.
4. On the **Create Maps Account** page, enter the following values:
 - The *Subscription* that you want to use for this account.
 - The *Resource group* name for this account. You may choose to *Create new* or *Use existing* resource group.
 - The *Name* of your new account.
 - The *Pricing tier* for this account.
 - Read the *License* and *Privacy Statement*, and check the checkbox to accept the terms.
 - Click the **Create** button.

Create Azure Maps Account - Microsoft Azure

https://ms.portal.azure.com/#create/MicrosoftMaps

Microsoft Azure (Preview)

Search resources, services, and docs (G+)

Home > New > Marketplace > Azure Maps >

Create Azure Maps Account

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use Resource groups like folders to organize and manage all your resources.

Subscription * contoso-subscription

Resource group * contoso-resource-group
Create new

ACCOUNT DETAILS

Name * contoso-maps

Pricing tier * Gen2 (Maps & Location Insights)
See pricing details and pricing tier guide

I confirm that I have read and agree to the [License and Privacy Statement](#). *

Note - Azure Maps shares customer-provided address/location queries ("Queries") with third party TomTom for mapping functionality purposes. Queries are not linked to any customer or end-user when shared with TomTom and cannot be used to identify individuals. Microsoft is currently in the process of adding TomTom to the Online Services Subcontractor List. Note that the Mobility and Weather Services which include integration with Moovit and AccuWeather are currently in PREVIEW.

Create

Get the primary key for your account

Once your Maps account is successfully created, retrieve the primary key that enables you to query the Maps APIs.

1. Open your Maps account in the portal.
2. In the settings section, select **Authentication**.
3. Copy the **Primary Key** to your clipboard. Save it locally to use later in this tutorial.

NOTE

If you use the subscription key instead of the primary key, your map won't render properly. Also, for security purposes, it is recommended that you rotate between your primary and secondary keys. To rotate keys, update your app to use the secondary key, deploy, then press the cycle/refresh button beside the primary key to generate a new primary key. The old primary key will be disabled. For more information on key rotation, see [Set up Azure Key Vault with key rotation and auditing](#)

The screenshot shows the Microsoft Azure (Preview) portal with the URL <https://ms.portal.azure.com/#@microsoft.onmicrosoft.com/resource/subscri...>. The page title is "contoso-maps | Authentication". The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Creator (PREVIEW), Creator overview, Settings (with Authentication selected), Pricing Tier, Properties, Locks, Export template, Help (Getting Started, About), and a search bar.

Azure Maps supports two ways to authenticate:

1. Azure Active Directory (Azure AD)(Preview) – Azure AD is Microsoft's cloud-based identity and access management service. Azure Maps Azure AD integration is currently available in preview for all Azure Maps API's. Azure AD supports role-based access control (RBAC) to allow fine-grained access to Azure Maps resources. To learn more about Azure Maps Azure AD integration, see [Azure Maps and Azure AD](#).
2. Shared Key Authentication – Shared Key authentication, often referred to as subscription key, relies on passing Azure Maps account generated keys with each request to Azure Maps. We recommend regenerating your keys regularly. You are provided two keys so that you can maintain connections using one key while regenerating the other. When you regenerate your keys, you must update any applications that access this account to use the new keys. To learn more about Azure Maps authentication, see [Authentication with Azure Maps](#).

Azure Active Directory Authentication

Client ID: <client id>

Shared Key Authentication

Primary Key: <primary key>

Secondary Key: <secondary key>

Download the demo application

1. Go to [interactiveSearch.html](#). Copy the content of the file.
2. Save the contents of this file locally as **AzureMapDemo.html**. Open it in a text editor.
3. Search for the string <Your Azure Maps Key>. Replace it with the **Primary Key** value from the preceding section.

Open the demo application

1. Open the file **AzureMapDemo.html** in a browser of your choice.
2. Observe the map shown of the City of Los Angeles. Zoom in and out to see how the map automatically renders with more or less information depending on the zoom level.
3. Change the default center of the map. In the **AzureMapDemo.html** file, search for the variable named **center**. Replace the longitude, latitude pair value for this variable with the new values **[-74.0060, 40.7128]**. Save the file and refresh your browser.
4. Try out the interactive search experience. In the search box on the upper-left corner of the demo web application, search for **restaurants**.
5. Move your mouse over the list of addresses and locations that appear below the search box. Notice how the corresponding pin on the map pops out information about that location. For privacy of private businesses, fictitious names and addresses are shown.

 **restaurant**

Brooklyn Deli & Pizza
38 Park Row, New York, NY 10038
phone: +(1)-(212)-9622833

Park Row Pizza
36 Park Row, New York, NY 10038
phone: +(1)-(212)-2272918
www.lunapizzanyc.com

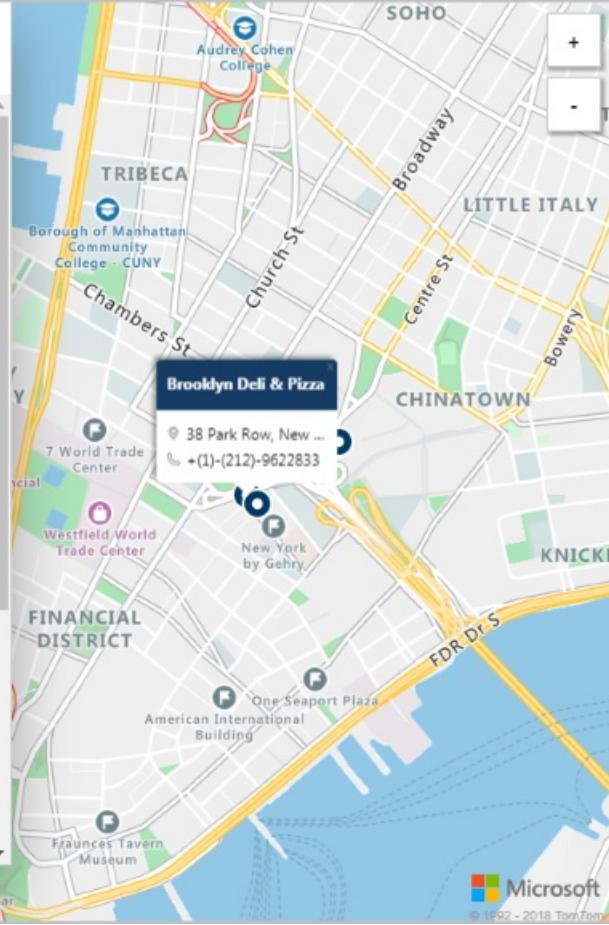
Woolworth Tower Kitchen
233 Broadway, New York, NY 10279
phone: +(1)-(212)-5712965
thewoolworthtowerkitchen.com

Two Rivers
10 Murray St, New York, NY 10007
phone: +(1)-(212)-5666915
www.tworiver.com

Muscle Maker Grill
10 Murray St, New York, NY 10007
phone: +(1)-(212)-9621813
www.ordermmq.com

Jin's Empire Asian Cuisine
10 Murray St, New York, NY 10007
phone: +(1)-(212)-3469688
www.jinsempireasian.com

Potbelly Sandwich Shop
280 Broadway, New York, NY 10007
phone: +(1)-(212)-6081462



The map shows the Lower Manhattan area with major streets like Broadway, Church St, Chambers St, Centre St, and Bowery. Key landmarks include the World Trade Center site, One Seaport Plaza, American International Building, and Fraunces Tavern Museum. A callout box highlights the location of Brooklyn Deli & Pizza at 38 Park Row, New York, NY 10038.

Clean up resources

WARNING

The tutorials listed in the [Next Steps](#) section detail how to use and configure Azure Maps with your account. Don't clean up the resources created in this quickstart if you plan to continue to the tutorials.

If you don't plan to continue to the tutorials, take these steps to clean up the resources:

1. Close the browser that runs the `AzureMapDemo.html` web application.
2. Navigate to the Azure portal page. Select **All resources** from the main portal page. Or, click on the menu icon in the upper left-hand corner. Select **All resources**.
3. Click on your Azure Maps account. At the top of the page, click **Delete**.

For more code examples and an interactive coding experience, see these guides:

[Find an address with Azure Maps search service](#)

[Use the Azure Maps Map Control](#)

Next steps

In this quickstart, you created your Azure Maps account and created a demo application. Take a look at the following tutorials to learn more about Azure Maps:

[Search nearby points of interest with Azure Maps](#)

Quickstart: Create an Android app with Azure Maps

4/30/2021 • 8 minutes to read • [Edit Online](#)

This article shows you how to add the Azure Maps to an Android app. It walks you through these basic steps:

- Setup your development environment.
- Create your own Azure Maps account.
- Get your primary Azure Maps key to use in the app.
- Reference the Azure Maps libraries from the project.
- Add an Azure Maps control to the app.

Prerequisites

1. Create an Azure Maps account by signing into the [Azure portal](#). If you don't have an Azure subscription, create a [free account](#) before you begin.
2. [Make an Azure Maps account](#)
3. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).
4. [Download Android Studio](#) for free from Google.

Create an Azure Maps account

Create a new Azure Maps account with the following steps:

1. In the upper left-hand corner of the [Azure portal](#), click **Create a resource**.
2. In the *Search the Marketplace* box, type **Azure Maps**.
3. From the *Results*, select **Azure Maps**. Click **Create** button that appears below the map.
4. On the **Create Maps Account** page, enter the following values:
 - The *Subscription* that you want to use for this account.
 - The *Resource group* name for this account. You may choose to *Create new* or *Use existing* resource group.
 - The *Name* of your new account.
 - The *Pricing tier* for this account.
 - Read the *License* and *Privacy Statement*, and check the checkbox to accept the terms.
 - Click the **Create** button.

Create Azure Maps Account - Microsoft Azure

https://ms.portal.azure.com/#create/MicrosoftMaps

Microsoft Azure (Preview)

Search resources, services, and docs (G+)

Home > New > Marketplace > Azure Maps >

Create Azure Maps Account

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use Resource groups like folders to organize and manage all your resources.

Subscription * contoso-subscription

Resource group * contoso-resource-group
Create new

ACCOUNT DETAILS

Name * contoso-maps

Pricing tier * Gen2 (Maps & Location Insights)
See pricing details and pricing tier guide

I confirm that I have read and agree to the [License and Privacy Statement](#). *

Note - Azure Maps shares customer-provided address/location queries ("Queries") with third party TomTom for mapping functionality purposes. Queries are not linked to any customer or end-user when shared with TomTom and cannot be used to identify individuals. Microsoft is currently in the process of adding TomTom to the Online Services Subcontractor List. Note that the Mobility and Weather Services which include integration with Moovit and AccuWeather are currently in PREVIEW.

Create

Get the primary key for your account

Once your Maps account is successfully created, retrieve the primary key that enables you to query the Maps APIs.

1. Open your Maps account in the portal.
2. In the settings section, select **Authentication**.
3. Copy the **Primary Key** to your clipboard. Save it locally to use later in this tutorial.

NOTE

If you use the Azure subscription key instead of the Azure Maps primary key, your map won't render properly. Also, for security purposes, it is recommended that you rotate between your primary and secondary keys. To rotate keys, update your app to use the secondary key, deploy, then press the cycle/refresh button beside the primary key to generate a new primary key. The old primary key will be disabled. For more information on key rotation, see [Set up Azure Key Vault with key rotation and auditing](#)

contoso-maps | Authentication

Azure Maps Account

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Events

Creator PREVIEW

Creator overview

Settings

Authentication

Pricing Tier

Properties

Locks

Export template

Help

Getting Started

About

Client ID <client id>

Azure Active Directory Authentication

Shared Key Authentication

Primary Key <primary key>

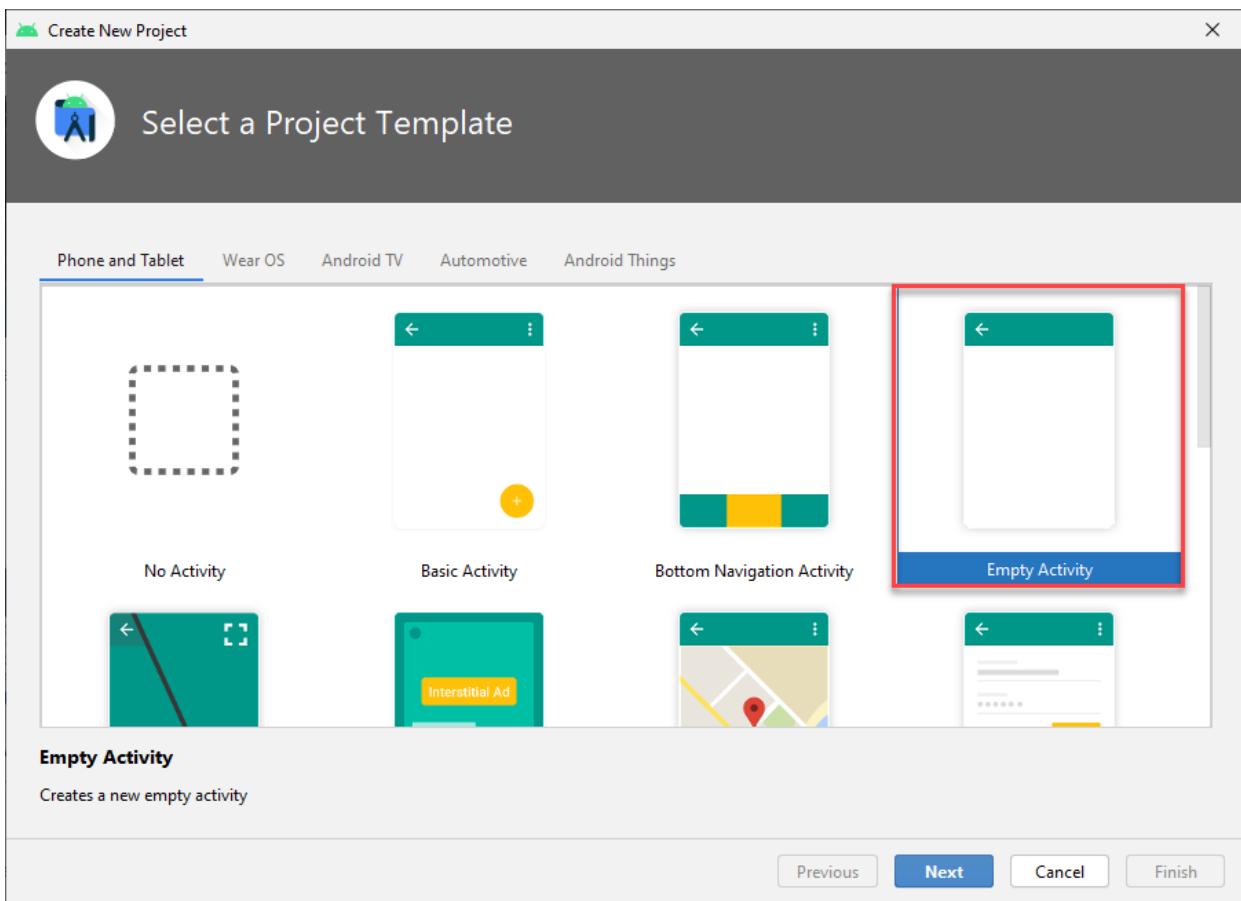
Secondary Key <secondary key>

Create a project in Android Studio

First, create a new project with an empty activity. Complete these steps to create an Android Studio project:

1. Under **Choose your project**, select **Phone and Tablet**. Your application will run on this form factor.
2. On the **Phone and Tablet** tab, select **Empty Activity**, and then select **Next**.
3. Under **Configure your project**, select **API 21: Android 5.0.0 (Lollipop)** as the minimum SDK. This is the earliest version supported by the Azure Maps Android SDK.
4. Accept the default **Activity Name** and **Layout Name** and select **Finish**.

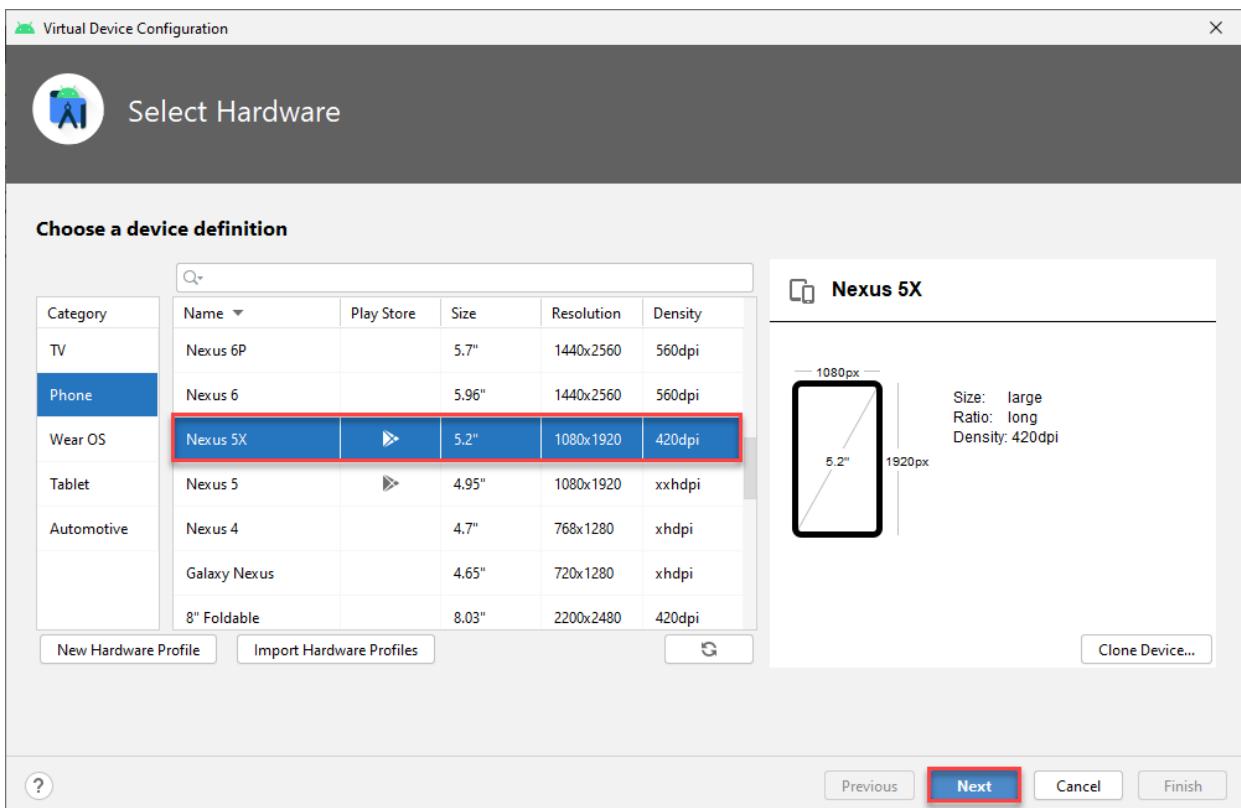
See the [Android Studio documentation](#) for more help with installing Android Studio and creating a new project.



Set up a virtual device

Android Studio lets you set up a virtual Android device on your computer. Doing so can help you test your application during development. To set up a virtual device, select the Android Virtual Device (AVD) Manager icon in the upper-right corner of your project screen, and then select **Create Virtual Device**. You can also get to the AVD Manager by selecting **Tools > Android > AVD Manager** from the toolbar. In the **Phones** category, select **Nexus 5X**, and then select **Next**.

You can learn more about setting up an AVD in the [Android Studio documentation](#).



Install the Azure Maps Android SDK

The next step in building your application is to install the Azure Maps Android SDK. Complete these steps to install the SDK:

1. Open the top-level `build.gradle` file and add the following code to the `all projects, repositories` block section:

```
maven {  
    url "https://atlas.microsoft.com/sdk/android"  
}
```

2. Update your `app/build.gradle` and add the following code to it:

- a. Make sure that your project's `minSdkVersion` is at API 21 or higher.
- b. Add the following code to the Android section:

```
compileOptions {  
    sourceCompatibility JavaVersion.VERSION_1_8  
    targetCompatibility JavaVersion.VERSION_1_8  
}
```

- c. Update your dependencies block and add a new implementation dependency line for the latest Azure Maps Android SDK:

```
implementation "com.microsoft.azure.maps:mapcontrol:0.7"
```

NOTE

You can set the version number to "0+" to have your code always point to the latest version.

d. Go to **File** in the toolbar and then click on **Sync Project with Gradle Files**.

3. Add a map fragment to the main activity (res > layout > activity_main.xml):

```
<com.microsoft.azure.maps.mapcontrol.MapControl  
    android:id="@+id/mapcontrol"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
/>
```

4. In the **MainActivity.java** file you'll need to:

- add imports for the Azure Maps SDK
- set your Azure Maps authentication information
- get the map control instance in the **onCreate** method

Setting the authentication information on the `AzureMaps` class globally using the `setSubscriptionKey` or `setAadProperties` methods makes it so you won't have to add your authentication information on every view.

The map control contains its own lifecycle methods for managing Android's OpenGL lifecycle. These lifecycle methods must be called directly from the containing Activity. For your app to correctly call the map control's lifecycle methods, you must override the following lifecycle methods in the Activity that contains the map control. And, you must call the respective map control method.

- `onCreate(Bundle)`
- `onDestroy()`
- `onLowMemory()`
- `onPause()`
- `onResume()`
- `onSaveInstanceState(Bundle)`
- `onStart()`
- `onStop()`

Edit the **MainActivity.java** file as follows:

```
package com.example.myapplication;  
  
import androidx.appcompat.app.AppCompatActivity;  
import com.microsoft.azure.maps.mapcontrol.AzureMaps;  
import com.microsoft.azure.maps.mapcontrol.MapControl;  
import com.microsoft.azure.maps.mapcontrol.layer.SymbolLayer;  
import com.microsoft.azure.maps.mapcontrol.options.MapStyle;  
import com.microsoft.azure.maps.mapcontrol.source.DataSource;  
  
public class MainActivity extends AppCompatActivity {  
  
    static {  
        AzureMaps.setSubscriptionKey("<Your Azure Maps subscription key>");  
  
        //Alternatively use Azure Active Directory authenticate.  
        //AzureMaps.setAadProperties("<Your aad clientId>", "<Your aad AppId>", "<Your aad Tenant>");  
    }  
  
    MapControl mapControl;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

```

        mapControl = findViewById(R.id.mapcontrol);

        mapControl.onCreate(savedInstanceState);

        //Wait until the map resources are ready.
        mapControl.onReady(map -> {
            //Add your post map load code here.

        });
    }

    @Override
    public void onResume() {
        super.onResume();
        mapControl.onResume();
    }

    @Override
    protected void onStart(){
        super.onStart();
        mapControl.onStart();
    }

    @Override
    public void onPause() {
        super.onPause();
        mapControl.onPause();
    }

    @Override
    public void onStop() {
        super.onStop();
        mapControl.onStop();
    }

    @Override
    public void onLowMemory() {
        super.onLowMemory();
        mapControl.onLowMemory();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        mapControl.onDestroy();
    }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        mapControl.onSaveInstanceState(outState);
    }
}

```

NOTE

After you complete the preceding steps, you may get warnings from Android Studio about some of the code. To resolve these warnings, import the classes referenced in `MainActivity.java`. You can automatically import these classes by selecting `Alt + Enter` (`Option + Return` on a Mac).

4. In the `MainActivity.kt` file you'll need to:

- add imports for the Azure Maps SDK
- set your Azure Maps authentication information

- get the map control instance in the `onCreate` method

Setting the authentication information on the `AzureMaps` class globally using the `setSubscriptionKey` or `setAadProperties` methods makes it so you won't have to add your authentication information on every view.

The map control contains its own lifecycle methods for managing Android's OpenGL lifecycle. These lifecycle methods must be called directly from the containing Activity. For your app to correctly call the map control's lifecycle methods, you must override the following lifecycle methods in the Activity that contains the map control. And, you must call the respective map control method.

- `onCreate(Bundle)`
- `onDestroy()`
- `onLowMemory()`
- `onPause()`
- `onResume()`
- `onSaveInstanceState(Bundle)`
- `onStart()`
- `onStop()`

Edit the `MainActivity.kt` file as follows:

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import com.microsoft.azure.maps.mapcontrol.AzureMap
import com.microsoft.azure.maps.mapcontrol.AzureMaps
import com.microsoft.azure.maps.mapcontrol.MapControl
import com.microsoft.azure.maps.mapcontrol.events.OnReady

class MainActivity : AppCompatActivity() {

    companion object {
        init {
            AzureMaps.setSubscriptionKey("<Your Azure Maps subscription key>");

            //Alternatively use Azure Active Directory authenticate.
            //AzureMaps.setAadProperties("<Your aad clientId>", "<Your aad AppId>", "<Your aad
Tenant>");
        }
    }

    var mapControl: MapControl? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        mapControl = findViewById(R.id.mapcontrol)

        mapControl?.onCreate(savedInstanceState)

        //Wait until the map resources are ready.
        mapControl?.onReady { map: AzureMap -> }
    }

    public override fun onStart() {
        super.onStart()
        mapControl?.onStart()
    }

    public override fun onResume() {
```

```
super.onResume()
mapControl?.onResume()
}

public override fun onPause() {
    mapControl?.onPause()
    super.onPause()
}

public override fun onStop() {
    mapControl?.onStop()
    super.onStop()
}

override fun onLowMemory() {
    mapControl?.onLowMemory()
    super.onLowMemory()
}

override fun onDestroy() {
    mapControl?.onDestroy()
    super.onDestroy()
}

override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    mapControl?.onSaveInstanceState(outState)
}
}
```

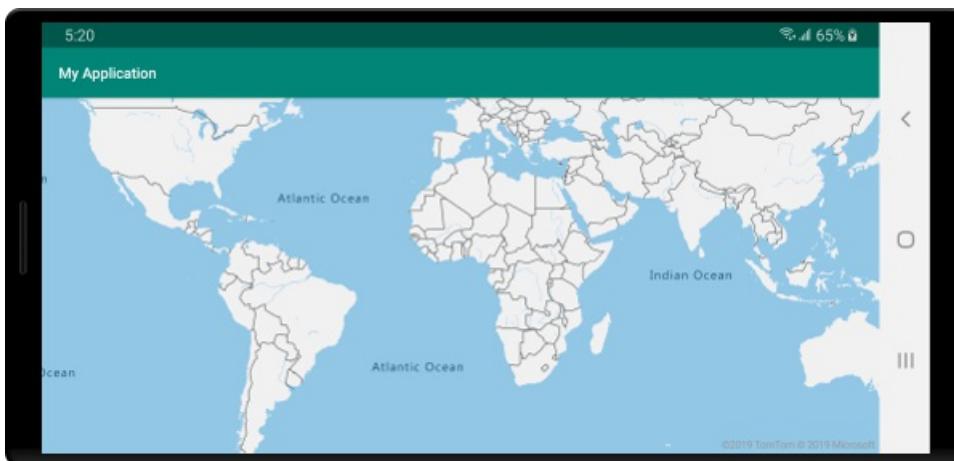
NOTE

After you complete the preceding steps, you may get warnings from Android Studio about some of the code. To resolve these warnings, import the classes referenced in `MainActivity.kt`. You can automatically import these classes by selecting `Alt + Enter` (`Option + Return` on a Mac).

5. Select the run button, as shown in the following graphic (or press `Control + R` on a Mac), to build your application.



Android Studio will take a few seconds to build the application. After the build is complete, you can test your application in the emulated Android device. You should see a map like this one:



Clean up resources

WARNING

The tutorials listed in the [Next Steps](#) section detail how to use and configure Azure Maps with your account. Don't clean up the resources created in this quickstart if you plan to continue to the tutorials.

If you don't plan to continue to the tutorials, take these steps to clean up the resources:

1. Close Android Studio and delete the application you created.
2. If you tested the application on an external device, uninstall the application from that device.

If you don't plan on continuing to develop with the Azure Maps Android SDK:

1. Navigate to the Azure portal page. Select **All resources** from the main portal page. Or, click on the menu icon in the upper left-hand corner. Select **All resources**.
2. Click on your Azure Maps account. At the top of the page, click **Delete**.
3. Optionally, if you don't plan to continue developing Android apps, uninstall Android Studio.

For more code examples, see these guides:

- [Manage authentication in Azure Maps](#)
- [Change map styles in Android maps](#)
- [Add a symbol layer](#)
- [Add a line layer](#)
- [Add a polygon layer](#)

Next steps

In this quickstart, you created your Azure Maps account and created a demo application. Take a look at the following tutorials to learn more about Azure Maps:

[Load GeoJSON data into Azure Maps](#)

Tutorial: Search nearby points of interest using Azure Maps

3/5/2021 • 6 minutes to read • [Edit Online](#)

This tutorial shows how to set up an account with Azure Maps, then use the Maps APIs to search for a point of interest. In this tutorial, you learn how to:

- Create an Azure Maps account
- Retrieve the primary key for your Maps account
- Create a new web page using the map control API
- Use the Maps search service to find a nearby point of interest

Prerequisites

1. Sign in to the [Azure portal](#). If you don't have an Azure subscription, create a [free account](#) before you begin.
2. [Make an Azure Maps account](#)
3. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

Create a new map

The Map Control API is a convenient client library. This API allows you to easily integrate Maps into your web application. It hides the complexity of the bare REST service calls and boosts your productivity with customizable components. The following steps show you how to create a static HTML page embedded with the Map Control API.

1. On your local machine, create a new file and name it **MapSearch.html**.
2. Add the following HTML components to the file:

```

<!DOCTYPE html>
<html>
<head>
    <title>Map Search</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet"
        href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css">
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <!-- Add a reference to the Azure Maps Services Module JavaScript file. -->
    <script src="https://atlas.microsoft.com/sdk/javascript/service/2/atlas-service.min.js"></script>
</script>

<script>
function GetMap(){
    //Add Map Control JavaScript code here.
}
</script>

<style>
    html,
    body {
        width: 100%;
        height: 100%;
        padding: 0;
        margin: 0;
    }

    #myMap {
        width: 100%;
        height: 100%;
    }
</style>
</head>
<body onload="GetMap()">
    <div id="myMap"></div>
</body>
</html>

```

Notice that the HTML header includes the CSS and JavaScript resource files hosted by the Azure Map Control library. Note the `onload` event on the body of the page, which will call the `GetMap` function when the body of the page has loaded. The `GetMap` function will contain the inline JavaScript code to access the Azure Maps APIs.

- Add the following JavaScript code to the `GetMap` function of the HTML file. Replace the string `<Your Azure Maps Key>` with the primary key that you copied from your Maps account.

```

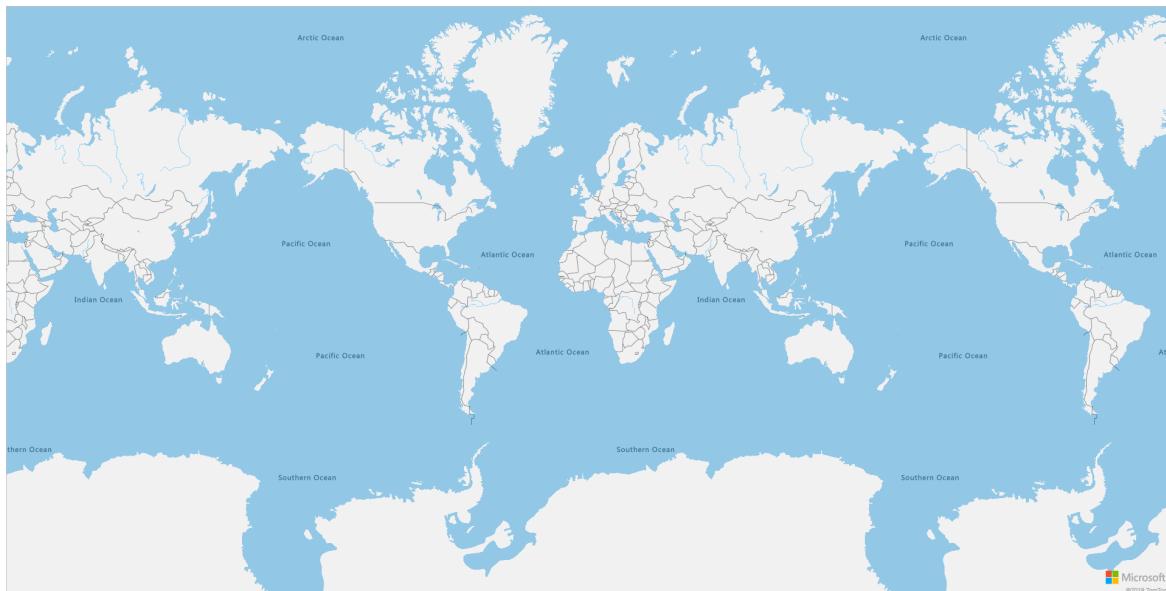
//Instantiate a map object
var map = new atlas.Map("myMap", {
    //Add your Azure Maps subscription key to the map SDK. Get an Azure Maps key at
    //https://azure.com/maps
    authOptions: {
        authType: 'subscriptionKey',
        subscriptionKey: '<Your Azure Maps Key>'
    }
});

```

This segment initializes the Map Control API for your Azure Maps account key. `atlas` is the namespace that contains the API and related visual components. `atlas.Map` provides the control for a visual and

interactive web map.

- Save your changes to the file and open the HTML page in a browser. The map shown is the most basic map that you can make by calling `atlas.Map` using your account key.



- In the `GetMap` function, after initializing the map, add the following JavaScript code.

```
//Wait until the map resources are ready.  
map.events.add('ready', function() {  
  
    //Create a data source and add it to the map.  
    datasource = new atlas.source.DataSource();  
    map.sources.add(datasource);  
  
    //Add a layer for rendering point data.  
    var resultLayer = new atlas.layer.SymbolLayer(datasource, null, {  
        iconOptions: {  
            image: 'pin-round-darkblue',  
            anchor: 'center',  
            allowOverlap: true  
        },  
        textOptions: {  
            anchor: "top"  
        }  
    });  
  
    map.layers.add(resultLayer);  
});
```

In this code segment, a `ready` event is added to the map, which will fire when the map resources have been loaded and the map is ready to be accessed. In the map `ready` event handler, a data source is created to store result data. A symbol layer is created and attached to the data source. This layer specifies how the result data in the data source should be rendered. In this case, the result is rendered with a dark blue round pin icon, centered over the results coordinate, and allows other icons to overlap. The result layer is added to the map layers.

Add search capabilities

This section shows how to use the Maps [Search API](#) to find a point of interest on your map. It's a RESTful API designed for developers to search for addresses, points of interest, and other geographical information. The Search service assigns a latitude and longitude information to a specified address. The [Service Module](#)

explained below can be used to search for a location using the Maps Search API.

Service Module

1. In the map `ready` event handler, construct the search service URL by adding the following JavaScript code.

```
// Use SubscriptionKeyCredential with a subscription key
var subscriptionKeyCredential = new
atlas.service.SubscriptionKeyCredential(atlas.getSubscriptionKey());

// Use subscriptionKeyCredential to create a pipeline
var pipeline = atlas.serviceMapsURL.newPipeline(subscriptionKeyCredential);

// Construct the SearchURL object
var searchURL = new atlas.service.SearchURL(pipeline);
```

The `SubscriptionKeyCredential` creates a `SubscriptionKeyCredentialPolicy` to authenticate HTTP requests to Azure Maps with the subscription key. The `atlas.serviceMapsURL.newPipeline()` takes in the `SubscriptionKeyCredential` policy and creates a `Pipeline` instance. The `searchURL` represents a URL to Azure Maps `Search` operations.

2. Next add the following script block to build the search query. It uses the Fuzzy Search Service, which is a basic search API of the Search Service. Fuzzy Search Service handles most fuzzy inputs like addresses, places, and points of interest (POI). This code searches for nearby Gasoline Stations within the specified radius of the provided latitude and longitude. A GeoJSON feature collection from the response is then extracted using the `geojson.getFeatures()` method and added to the data source, which automatically results in the data being rendered on the map via the symbol layer. The last part of the script sets the maps camera view using the bounding box of the results using the Map's `setCamera` property.

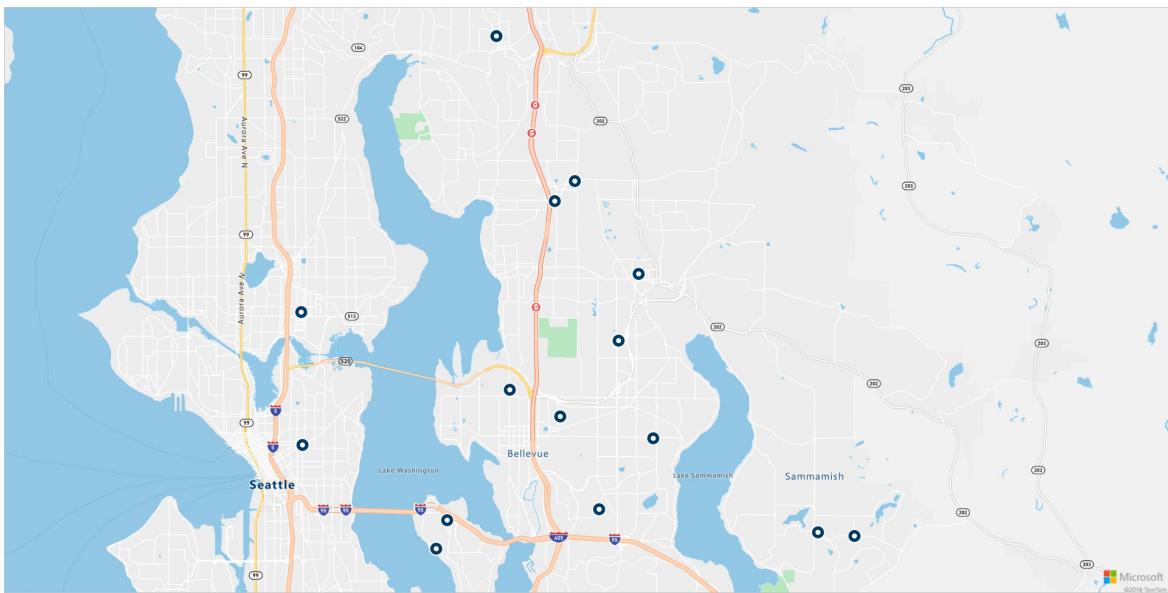
```
var query = 'gasoline-station';
var radius = 9000;
var lat = 47.64452336193245;
var lon = -122.13687658309935;

searchURL.searchPOI(atlas.serviceAborter.timeout(10000), query, {
    limit: 10,
    lat: lat,
    lon: lon,
    radius: radius
}).then((results) => {

    // Extract GeoJSON feature collection from the response and add it to the datasource
    var data = results.geojson.getFeatures();
    datasource.add(data);

    // set camera to bounds to show the results
    map.setCamera({
        bounds: data.bbox,
        zoom: 10
    });
});
```

3. Save the `MapSearch.html` file and refresh your browser. You should see the map centered on Seattle with round-blue pins for locations of gasoline stations in the area.



4. You can see the raw data that the map is rendering by entering the following `HTTPRequest` in your browser. Replace <Your Azure Maps Key> with your primary key.

```
https://atlas.microsoft.com/search/poi/json?api-version=1.0&query=gasoline%20station&subscription-key=<subscription-key>&lat=47.6292&lon=-122.2337&radius=100000
```

At this point, the MapSearch page can display the locations of points of interest that are returned from a fuzzy search query. Let's add some interactive capabilities and more information about the locations.

Add interactive data

The map that we've made so far only looks at the longitude/latitude data for the search results. However, the raw JSON that the Maps Search service returns contains additional information about each gas station. Including the name and street address. You can incorporate that data into the map with interactive popup boxes.

1. Add the following lines of code in the map `ready` event handler after the code to query the fuzzy search service. This code will create an instance of a Popup and add a mouseover event to the symbol layer.

```
//Create a popup but leave it closed so we can update it and display it later.  
popup = new atlas.Popup();  
  
//Add a mouse over event to the result layer and display a popup when this event fires.  
map.events.add('mouseover', resultLayer, showPopup);
```

The API `*atlas.Popup` provides an information window anchored at the required position on the map.

2. Add the following code within the `GetMap` function, to show the moused over result information in the popup.

```

function showPopup(e) {
    //Get the properties and coordinates of the first shape that the event occurred on.

    var p = e.shapes[0].getProperties();
    var position = e.shapes[0].getCoordinates();

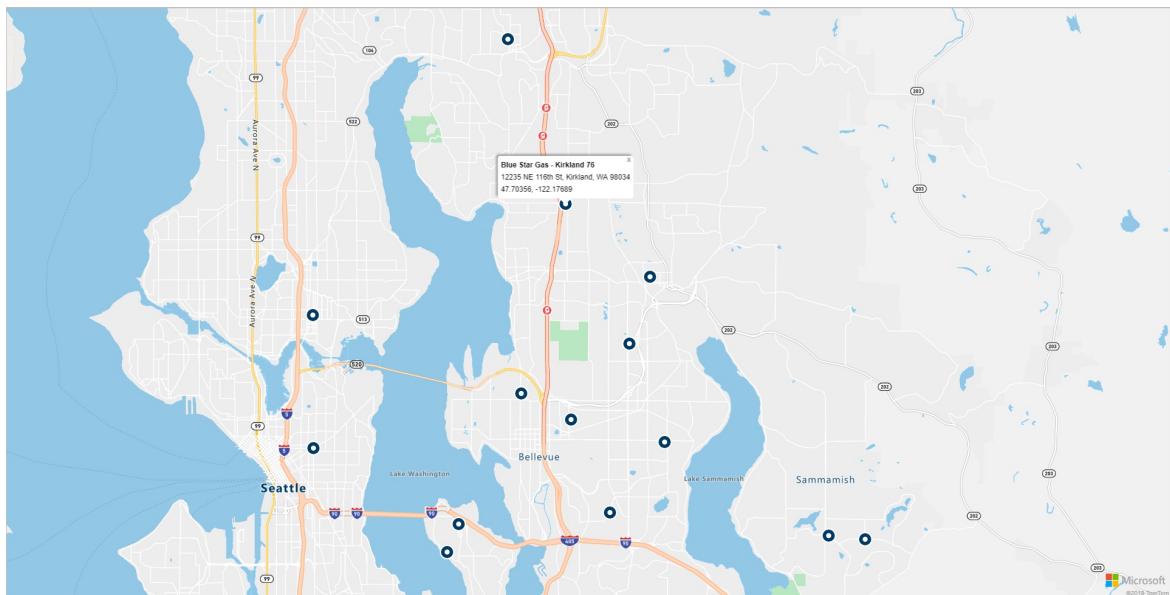
    //Create HTML from properties of the selected result.
    var html = `
        <div style="padding:5px">
            <div><b>${p.poi.name}</b></div>
            <div>${p.address.freeformAddress}</div>
            <div>${position[1]}, ${position[0]}</div>
        </div>`;

    //Update the content and position of the popup.
    popup.setPopupOptions({
        content: html,
        position: position
    });

    //Open the popup.
    popup.open(map);
}

```

- Save the file and refresh your browser. Now the map in the browser shows information pop-ups when you hover over any of the search pins.



To view the full code for this tutorial, click [here](#). To view the live sample, click [here](#)

Clean up resources

There are no resources that require cleanup.

Next steps

The next tutorial demonstrates how to display a route between two locations.

[Route to a destination](#)

Tutorial: How to display route directions using Azure Maps Route service and Map control

4/30/2021 • 6 minutes to read • [Edit Online](#)

This tutorial shows you how to use the Azure Maps [Route service API](#) and [Map control](#) to display route directions from start to end point. In this tutorial, you'll learn how to:

- Create and display the Map control on a web page.
- Define the display rendering of the route by defining [Symbol layers](#) and [Line layers](#).
- Create and add GeoJSON objects to the Map to represent start and end points.
- Get route directions from start and end points using the [Get Route directions API](#).

You can obtain the full source code for the sample [here](#). A live sample can be found [here](#).

Prerequisites

1. [Make an Azure Maps account](#)
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.

Create and display the Map control

The following steps show you how to create and display the Map control in a web page.

1. On your local machine, create a new file and name it **MapRoute.html**.
2. Copy/paste the following HTML markup into the file.

```

<!DOCTYPE html>
<html>
<head>
    <title>Map Route</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet"
        href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css">
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <!-- Add a reference to the Azure Maps Services Module JavaScript file. -->
    <script src="https://atlas.microsoft.com/sdk/javascript/service/2/atlas-service.min.js"></script>

    <script>
        var map, datasource, client;

        function GetMap() {
            //Add Map Control JavaScript code here.
        }
    </script>
    <style>
        html,
        body {
            width: 100%;
            height: 100%;
            padding: 0;
            margin: 0;
        }

        #myMap {
            width: 100%;
            height: 100%;
        }
    </style>
</head>
<body onload="GetMap()">
    <div id="myMap"></div>
</body>
</html>

```

The HTML header includes the CSS and JavaScript resource files hosted by the Azure Map Control library. The body's `onload` event calls the `GetMap` function. In the next step, we'll add the Map control initialization code.

3. Add the following JavaScript code to the `GetMap` function. Replace the string `<Your Azure Maps Key>` with the primary key that you copied from your Maps account.

```

//Instantiate a map object
var map = new atlas.Map("myMap", {
    //Add your Azure Maps subscription key to the map SDK. Get an Azure Maps key at
    //https://azure.com/maps
    authOptions: {
        authType: 'subscriptionKey',
        subscriptionKey: '<Your Azure Maps Key>'
    }
});

```

4. Save the file and open it in your browser. A simple is displayed.



Define route display rendering

In this tutorial, we'll render the route using a line layer. The start and end points will be rendered using a symbol layer. For more information on adding line layers, see [Add a line layer to a map](#). To learn more about symbol layers, see [Add a symbol layer to a map](#).

1. Append the following JavaScript code in the `GetMap` function. This code implements the Map control's `ready` event handler. The rest of the code in this tutorial will be placed inside the `ready` event handler.

```
//Wait until the map resources are ready.  
map.events.add('ready', function() {  
  
    //Create a data source and add it to the map.  
    datasource = new atlas.source.DataSource();  
    map.sources.add(datasource);  
  
    //Add a layer for rendering the route lines and have it render under the map labels.  
    map.layers.add(new atlas.layer.LineLayer(datasource, null, {  
        strokeColor: '#2272B9',  
        strokeWidth: 5,  
        lineJoin: 'round',  
        lineCap: 'round'  
    }), 'labels');  
  
    //Add a layer for rendering point data.  
    map.layers.add(new atlas.layer.SymbolLayer(datasource, null, {  
        iconOptions: {  
            image: ['get', 'icon'],  
            allowOverlap: true  
        },  
        textOptions: {  
            textField: ['get', 'title'],  
            offset: [0, 1.2]  
        },  
        filter: ['any', ['==', ['geometry-type'], 'Point'], ['==', ['geometry-type'], 'MultiPoint']]  
    //Only render Point or MultiPoints in this layer.  
    }));  
});
```

In the map control's `ready` event handler, a data source is created to store the route from start to end point. To define how the route line will be rendered, a line layer is created and attached to the data source. To ensure that the route line doesn't cover up the road labels, we've passed a second parameter with the value of `'labels'`.

Next, a symbol layer is created and attached to the data source. This layer specifies how the start and end points are rendered. Expressions have been added to retrieve the icon image and text label information from properties on each point object. To learn more about expressions, see [Data-driven style expressions](#).

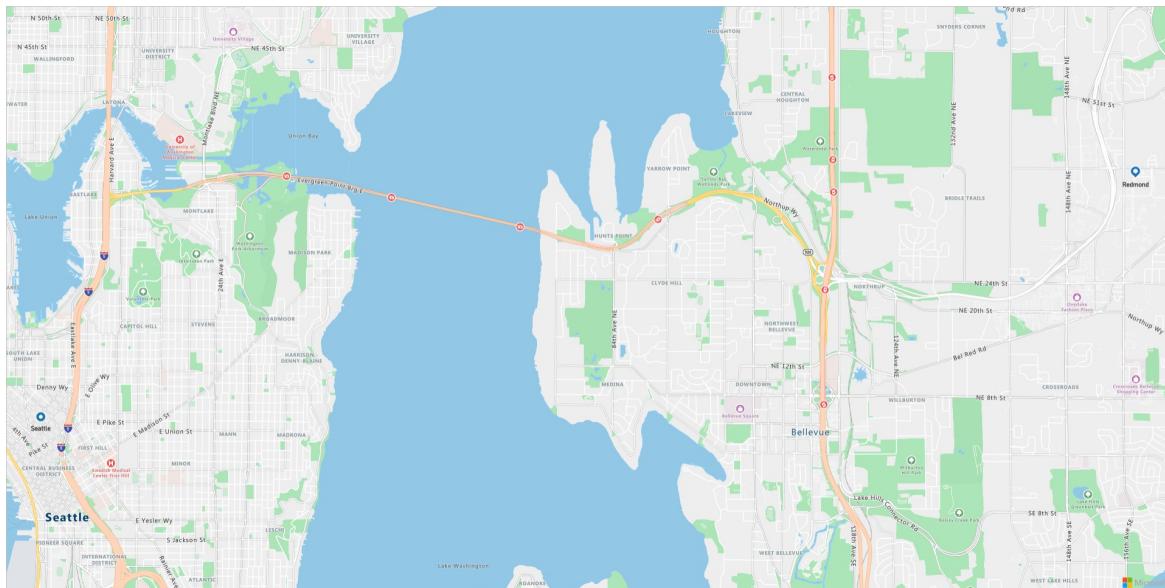
- Set the start point as Microsoft, and the end point as a gas station in Seattle. In the Map control's `ready` event handler, append the following code.

```
//Create the GeoJSON objects which represent the start and end points of the route.  
var startPoint = new atlas.data.Feature(new atlas.data.Point([-122.130137, 47.644702]), {  
    title: "Redmond",  
    icon: "pin-blue"  
});  
  
var endPoint = new atlas.data.Feature(new atlas.data.Point([-122.3352, 47.61397]), {  
    title: "Seattle",  
    icon: "pin-round-blue"  
});  
  
//Add the data to the data source.  
datasource.add([startPoint, endPoint]);  
  
map.setCamera({  
    bounds: atlas.data.BoundingBox.fromData([startPoint, endPoint]),  
    padding: 80  
});
```

This code creates two [GeoJSON Point objects](#) to represent start and end points, which are then added to the data source.

The last block of code sets the camera view using the latitude and longitude of the start and end points. The start and end points are added to the data source. The bounding box for the start and end points is calculated using the `atlas.data.BoundingBox.fromData` function. This bounding box is used to set the map cameras view over the entire route using the `map.setCamera` function. Padding is added to compensate for the pixel dimensions of the symbol icons. For more information about the Map control's `setCamera` property, see [setCamera\(CameraOptions | CameraBoundsOptions & AnimationOptions\)](#) property.

- Save `MapRoute.html` and refresh your browser. The map is now centered over Seattle. The teardrop blue pin marks the start point. The round blue pin marks the end point.



Get route directions

This section shows you how to use the Azure Maps Route Directions API to get route directions and the estimated time of arrival from one point to another.

TIP

The Azure Maps Route services offer APIs to plan routes based on different route types such as *fastest*, *shortest*, *eco*, or *thrilling* routes based on distance, traffic conditions, and mode of transport used. The service also lets users plan future routes based on historical traffic conditions. Users can see the prediction of route durations for any given time. For more information, see [Get Route directions API](#).

1. In the `GetMap` function, inside the control's `ready` event handler, add the following to the JavaScript code.

```
// Use SubscriptionKeyCredential with a subscription key
var subscriptionKeyCredential = new
atlas.service.SubscriptionKeyCredential(atlas.getSubscriptionKey());

// Use subscriptionKeyCredential to create a pipeline
var pipeline = atlas.serviceMapsURL.newPipeline(subscriptionKeyCredential);

// Construct the RouteURL object
var routeURL = new atlas.service.RouteURL(pipeline);
```

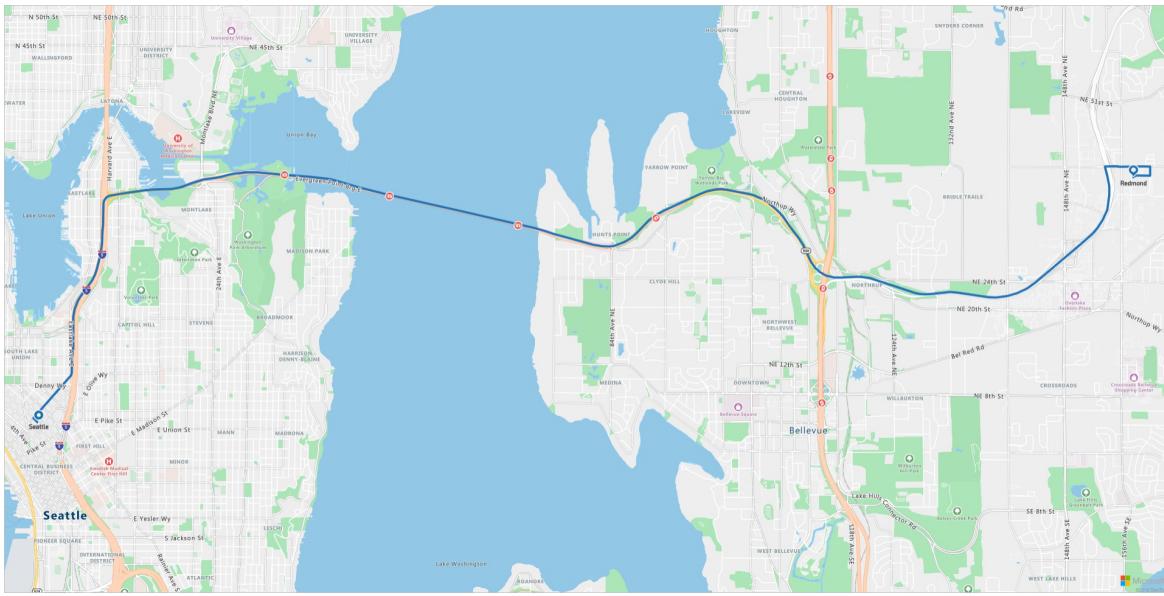
The `SubscriptionKeyCredential` creates a `SubscriptionKeyCredentialPolicy` to authenticate HTTP requests to Azure Maps with the subscription key. The `atlas.serviceMapsURL.newPipeline()` takes in the `SubscriptionKeyCredential` policy and creates a `Pipeline` instance. The `routeURL` represents a URL to Azure Maps [Route](#) operations.

2. After setting up credentials and the URL, append the following code in the control's `ready` event handler. This code constructs the route from start point to end point. The `routeURL` requests the Azure Maps Route service API to calculate route directions. A GeoJSON feature collection from the response is then extracted using the `geojson.getFeatures()` method and added to the data source.

```
//Start and end point input to the routeURL
var coordinates= [[startPoint.geometry.coordinates[0], startPoint.geometry.coordinates[1]],
[ endPoint.geometry.coordinates[0], endPoint.geometry.coordinates[1]]];

//Make a search route request
routeURL.calculateRouteDirections(atlas.serviceAborter.timeout(10000),
coordinates).then((directions) => {
    //Get data features from response
    var data = directions.geojson.getFeatures();
    datasource.add(data);
});
```

3. Save the **MapRoute.html** file and refresh your web browser. The map should now display the route from start to end point.



You can obtain the full source code for the sample [here](#). A live sample can be found [here](#).

Clean up resources

There are no resources that require cleanup.

Next steps

The next tutorial shows you how to create a route query with restrictions, like mode of travel or type of cargo. You can then display multiple routes on the same map.

[Find routes for different modes of travel](#)

Tutorial: Find and display routes for different modes of travel using Azure Maps

3/5/2021 • 9 minutes to read • [Edit Online](#)

This tutorial shows you how to use the Azure Maps [Route service](#) and [Map control](#) to display route directions for both private vehicles and commercial vehicles (trucks) with `USHAZMATClass2` cargo type . In addition, we'll walk you through how to visualize real-time traffic data on a map. In this tutorial, you learn how to:

- Create and display the Map control on a web page
- Render real-time traffic data on a map
- Request and display private and commercial vehicle routes on a map

Prerequisites

1. Sign in to the [Azure portal](#).
2. [Create an Azure Maps account](#).
3. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

You can obtain the full source code for the sample [here](#). A live sample can be found [here](#).

Create a new web page using the map control API

The following steps show you how to create and display the Map control in a web page.

1. On your local machine, create a new file and name it `MapTruckRoute.html`.
2. Copy/paste the following HTML markup into the file.

```

<!DOCTYPE html>
<html>
<head>
    <title>Map Route</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet"
        href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css">
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <!-- Add a reference to the Azure Maps Services Module JavaScript file. -->
    <script src="https://atlas.microsoft.com/sdk/javascript/service/2/atlas-service.min.js"></script>

    <script>
        var map, datasource, client;

        function GetMap() {
            //Add Map Control JavaScript code here.
        }
    </script>
    <style>
        html,
        body {
            width: 100%;
            height: 100%;
            padding: 0;
            margin: 0;
        }

        #myMap {
            width: 100%;
            height: 100%;
        }
    </style>
</head>
<body onload="GetMap()">
    <div id="myMap"></div>
</body>
</html>

```

The HTML header includes the CSS and JavaScript resource files hosted by the Azure Map Control library. The body's `onload` event calls the `GetMap` function. In the next step, we'll add the Map control initialization code.

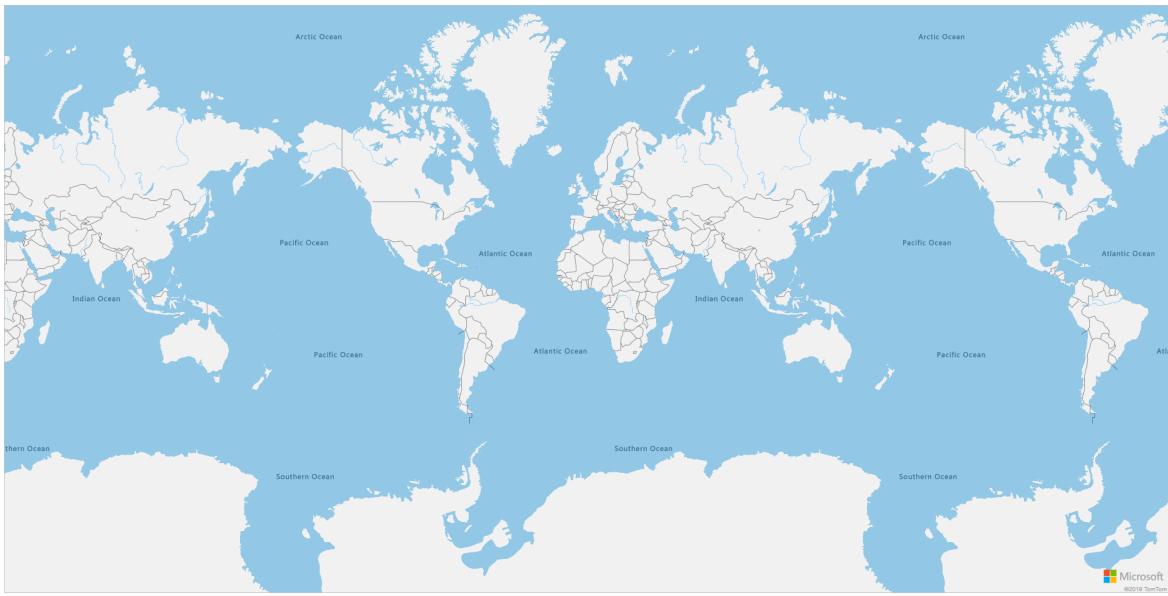
3. Add the following JavaScript code to the `GetMap` function. Replace the string `<Your Azure Maps Key>` with the primary key that you copied from your Maps account.

```

//Instantiate a map object
var map = new atlas.Map("myMap", {
    //Add your Azure Maps subscription key to the map SDK. Get an Azure Maps key at
    //https://azure.com/maps
    authOptions: {
        authType: 'subscriptionKey',
        subscriptionKey: '<Your Azure Maps Key>'
    }
});

```

4. Save the file and open it in your browser. A simple is displayed.



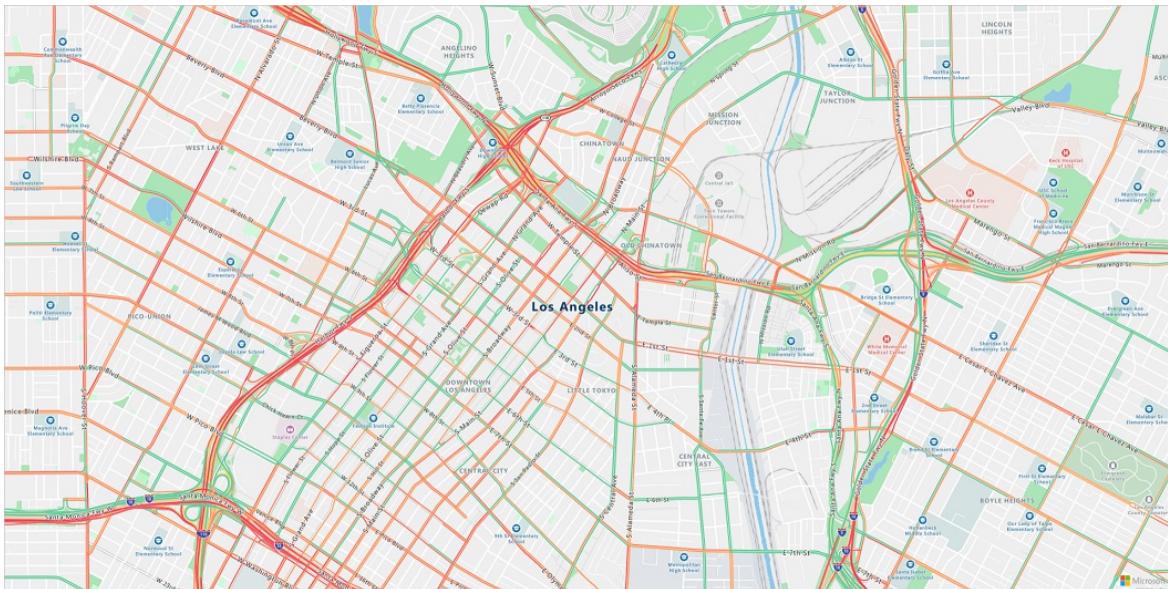
Render real-time traffic data on a map

1. Append the following JavaScript code in the `GetMap` function. This code implements the Map control's `ready` event handler. The rest of the code in this tutorial will be placed inside the `ready` event handler.

```
map.events.add("ready", function() {
    // Add Traffic Flow to the Map
    map.setTraffic({
        flow: "relative"
    });
});
```

In the map `ready` event handler, the traffic flow setting on the map is set to `relative`, which is the speed of the road relative to free-flow. For more traffic options, see [TrafficOptions interface](#).

2. Save the `MapTruckRoute.html` file and refresh the page in your browser. If you zoom into any city, like Los Angeles, you will see that the streets display with current traffic flow data.



Define route display rendering

In this tutorial, two routes will be calculated and rendered on the map. The first route will be calculated for a

private vehicle (car). The second route will be calculated for a commercial vehicle (truck) to show the difference between the results. When rendered, the map will display a symbol icon for the start and end points of the route, and route line geometries with different colors for each route path. For more information on adding line layers, see [Add a line layer to a map](#). To learn more about symbol layers, see [Add a symbol layer to a map](#).

1. In the Map control's `ready` event handler, append the following code.

```
//Create a data source and add it to the map.  
datasource = new atlas.source.DataSource();  
map.sources.add(datasource);  
  
//Add a layer for rendering the route lines and have it render under the map labels.  
map.layers.add(new atlas.layer.LineLayer(datasource, null, {  
    strokeColor: ['get', 'strokeColor'],  
    strokeWidth: ['get', 'strokeWidth'],  
    lineJoin: 'round',  
    lineCap: 'round'  
}), 'labels');  
  
//Add a layer for rendering point data.  
map.layers.add(new atlas.layer.SymbolLayer(datasource, null, {  
    iconOptions: {  
        image: ['get', 'icon'],  
        allowOverlap: true  
    },  
    textOptions: {  
        textField: ['get', 'title'],  
        offset: [0, 1.2]  
    },  
    filter: ['any', ['==', ['geometry-type'], 'Point'], ['==', ['geometry-type'], 'MultiPoint']]  
//Only render Point or MultiPoints in this layer.  
}));
```

In the Map control's `ready` event handler, a data source is created to store the route from start to finish. [Expressions](#) are used to retrieve the line width and color from properties on the route line feature. To ensure that the route line doesn't cover up the road labels, we've passed a second parameter with the value of `'labels'`.

Next, a symbol layer is created and attached to the data source. This layer specifies how the start and end points are rendered. Expressions have been added to retrieve the icon image and text label information from properties on each point object. To learn more about expressions, see [Data-driven style expressions](#).

2. Set the start point as a fictitious company in Seattle called Fabrikam, and the end point as a Microsoft office. In the Map control's `ready` event handler, append the following code.

```

//Create the GeoJSON objects which represent the start and end point of the route.
var startPoint = new atlas.data.Feature(new atlas.data.Point([-122.356099, 47.580045]), {
    title: 'Fabrikam, Inc.',
    icon: 'pin-blue'
});

var endPoint = new atlas.data.Feature(new atlas.data.Point([-122.201164, 47.616940]), {
    title: 'Microsoft - Lincoln Square',
    icon: 'pin-round-blue'
});

//Add the data to the data source.
datasource.add([startPoint, endPoint]);

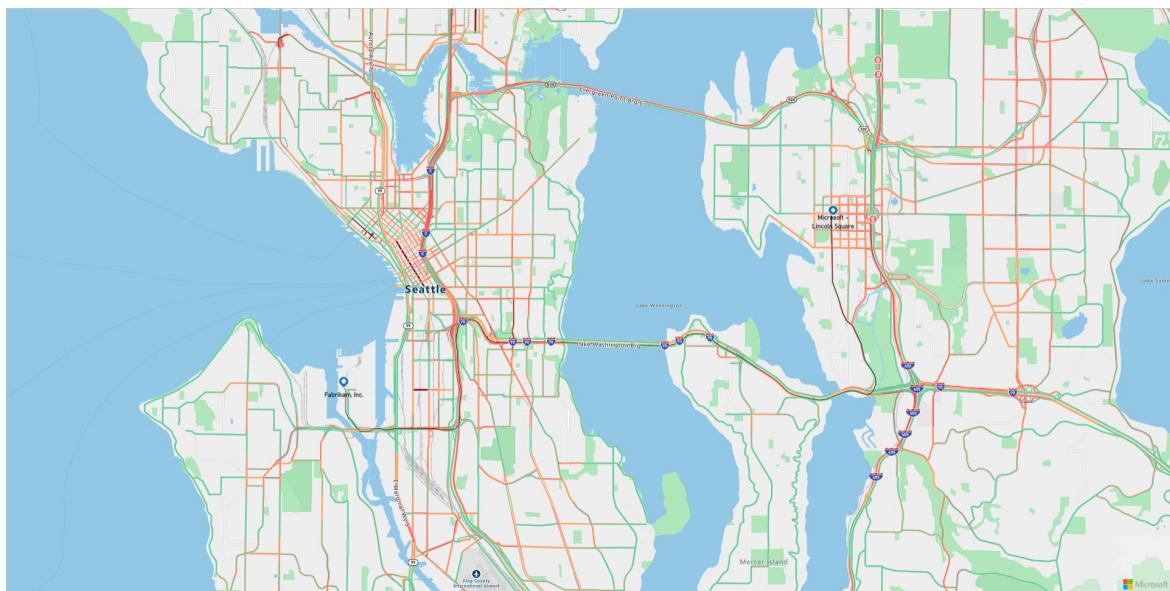
//Fit the map window to the bounding box defined by the start and end positions.
map.setCamera({
    bounds: atlas.data.BoundingBox.fromData([startPoint, endPoint]),
    padding: 100
});

```

This code creates two [GeoJSON Point objects](#) to represent start and end points, which are then added to the data source.

The last block of code sets the camera view using the latitude and longitude of the start and end points. The start and end points are added to the data source. The bounding box for the start and end points is calculated using the `atlas.data.BoundingBox.fromData` function. This bounding box is used to set the map cameras view over the entire route using the `map.setCamera` function. Padding is added to compensate for the pixel dimensions of the symbol icons. For more information about the Map control's `setCamera` property, see [setCamera\(CameraOptions | CameraBoundsOptions & AnimationOptions\)](#) property.

3. Save `TruckRoute.html` and refresh your browser. The map is now centered over Seattle. The teardrop blue pin marks the start point. The round blue pin marks the end point.



Request and display private and commercial vehicle routes on a map

This section shows you how to use the Azure Maps Route service to get directions from one point to another, based on your mode of transport. We'll be using two modes of transport: truck and car.

TIP

The Route service provides APIs to plan *fastest*, *shortest*, *eco*, or *thrilling* routes based on distance, traffic conditions, and mode of transport used. The service also lets users plan future routes based on historical traffic conditions. Users can see the prediction of route durations for any given time. For more information, see [Get Route directions API](#).

1. In the `GetMap` function, inside the control's `ready` event handler, add the following to the JavaScript code.

```
// Use SubscriptionKeyCredential with a subscription key
var subscriptionKeyCredential = new
atlas.service.SubscriptionKeyCredential(atlas.getSubscriptionKey());

// Use subscriptionKeyCredential to create a pipeline
var pipeline = atlas.serviceMapsURL.newPipeline(subscriptionKeyCredential);

// Construct the RouteURL object
var routeURL = new atlas.service.RouteURL(pipeline);
```

The `SubscriptionKeyCredential` creates a `SubscriptionKeyCredentialPolicy` to authenticate HTTP requests to Azure Maps with the subscription key. The `atlas.serviceMapsURL.newPipeline()` takes in the `SubscriptionKeyCredential` policy and creates a `Pipeline` instance. The `routeURL` represents a URL to Azure Maps [Route](#) operations.

2. After setting up credentials and the URL, add the following JavaScript code to construct a truck route route from start to end point. This route is created and displayed for a truck carrying `USHazmatClass2` classed cargo.

```
//Start and end point input to the routeURL
var coordinates= [[startPoint.geometry.coordinates[0], startPoint.geometry.coordinates[1]],
[ endPoint.geometry.coordinates[0], endPoint.geometry.coordinates[1]]];

//Make a search route request for a truck vehicle type
routeURL.calculateRouteDirections(atlas.serviceAborter.timeout(10000), coordinates,{
    travelMode: 'truck',
    vehicleWidth: 2,
    vehicleHeight: 2,
    vehicleLength: 5,
    vehicleLoadType: 'USHazmatClass2'
}).then((directions) => {
    //Get data features from response
    var data = directions.geojson.getFeatures();

    //Get the route line and add some style properties to it.
    var routeLine = data.features[0];
    routeLine.properties.strokeColor = '#2272B9';
    routeLine.properties.strokeWidth = 9;

    //Add the route line to the data source. We want this to render below the car route which will
    likely be added to the data source faster, so insert it at index 0.
    datasource.add(routeLine, 0);
});
```

The code above queries the Azure Maps Route service through the [Azure Maps Route Directions API](#). The route line is then extracted from the GeoJSON feature collection from the response that is extracted using the `geojson.getFeatures()` method. Finally, the route line is added to the data source. We are adding it at the index of 0, to ensure that the truck route is rendered before any other lines in the data source, because the truck route calculation will often be slower than a car route calculation. If the truck route line

is added to the data source after the car route, it will render above it. Two properties are added to the truck route line: a blue stroke color, and a stroke width of nine pixels.

TIP

To see all possible options and values for the Azure Maps Route Directions API, see [URI Parameters for Post Route Directions](#).

- Now append the following JavaScript code to construct a route for a car.

```
routeURL.calculateRouteDirections(atlas.service.Aborter.timeout(10000),
coordinates).then((directions) => {

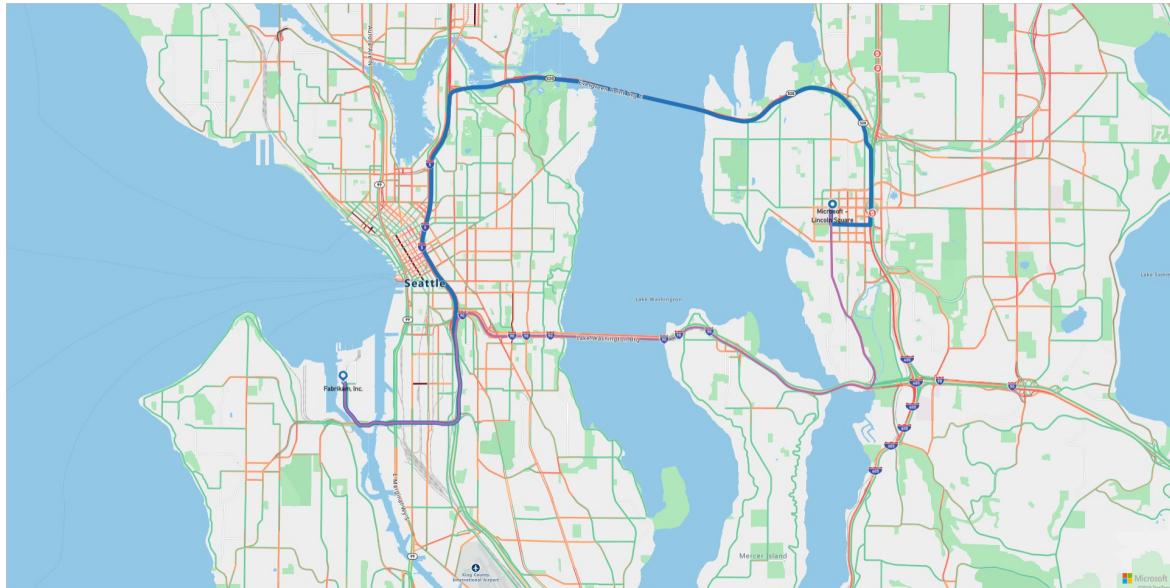
    //Get data features from response
    var data = directions.geojson.getFeatures();

    //Get the route line and add some style properties to it.
    var routeLine = data.features[0];
    routeLine.properties.strokeColor = '#B76DAB';
    routeLine.properties.strokeWidth = 5;

    //Add the route line to the data source. This will add the car route after the truck route.
    datasource.add(routeLine);
});
```

The code above queries the Azure Maps routing service through the [Azure Maps Route Directions API](#) method. The route line is then extracted from the GeoJSON feature collection from the response that is extracted using the `geojson.getFeatures()` method. Finally, the route line is added to the data source. Two properties are added to the truck route line: a purple stroke color, and a stroke width of five pixels.

- Save the `TruckRoute.html` file and refresh your web browser. The map should now display the truck and car routes.



The truck route is displayed using a thick blue line. The car route is displayed using a thin purple line. The car route goes across Lake Washington via I-90, passing through tunnels beneath residential areas. Because the tunnels are close to residential areas, hazardous waste cargo is restricted. The truck route, which specifies a `USHazmatClass2` cargo type, is directed to use a different highway.

You can obtain the full source code for the sample [here](#). A live sample can be found [here](#).

You can also [Use data-driven style expressions](#)

Clean up resources

There are no resources that require cleanup.

Next steps

The next tutorial demonstrates the process of creating a simple store locator by using Azure Maps.

[Create a store locator using Azure Maps](#)

Tutorial: Use Azure Maps to create a store locator

6/8/2021 • 23 minutes to read • [Edit Online](#)

This tutorial guides you through the process of creating a simple store locator using Azure Maps. In this tutorial, you'll learn how to:

- Create a new webpage by using the Azure Map Control API.
- Load custom data from a file and display it on a map.
- Use the Azure Maps Search service to find an address or enter a query.
- Get the user's location from the browser and show it on the map.
- Combine multiple layers to create custom symbols on the map.
- Cluster data points.
- Add zoom controls to the map.

Prerequisites

1. [Make an Azure Maps account in Gen 1 \(S1\) or Gen 2 pricing tier.](#)
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.

For more information about Azure Maps authentication, see [manage authentication in Azure Maps](#).

This tutorial uses the [Visual Studio Code](#) application, but you can use a different coding environment.

Sample code

In this tutorial, we'll create a store locator for a fictional company called Contoso Coffee. Also, the tutorial includes some tips to help you learn about extending the store locator with other optional functionalities.

You can view the [Live store locator sample here](#).

To more easily follow and engage this tutorial, you'll need to download the following resources:

- [Full source code for simple store locator sample](#)
- [Store location data to import into the store locator dataset](#)
- [Map images](#)

Store locator features

This section lists the features that are supported in the Contoso Coffee store locator application.

User interface features

- Store logo on the header
- Map supports panning and zooming
- A My Location button to search over the user's current location.
- Page layout adjusts based on the width of the device screen
- A search box and a search button

Functionality features

- A `keypress` event added to the search box triggers a search when the user presses `Enter`.
- When the map moves, the distance to each location from the center of the map calculates. The results list

updates to display the closest locations at the top of the map.

- When the user selects a result in the results list, the map is centered over the selected location and information about the location appears in a pop-up window.
- When the user selects a specific location, the map triggers a pop-up window.
- When the user zooms out, locations are grouped in clusters. Each cluster is represented by a circle with a number inside the circle. Clusters form and separate as the user changes the zoom level.
- Selecting a cluster zooms in two levels on the map and centers over the location of the cluster.

Store locator design

The following figure shows a wireframe of the general layout of our store locator. You can view the live wireframe [here](#).

The wireframe displays a map of the Bellevue area, Washington, showing various neighborhoods like Feriton, Yarrow Point, Ashwood, Belridge, and West Bellevue. A specific location, "14320 NE 20th St, Bellevue, WA 98007", is highlighted with a blue callout box. The sidebar on the left lists five store locations with their addresses, opening times, and distances from the user's current location (98007). The Microsoft logo and a copyright notice for TomTom are visible in the bottom right corner of the map.

Location	Address	City	Zip Code	Open until	Distance
1299 156th Ave NE	Bellevue, WA 98007	Bellevue	98007	18:00 PM	0.51 miles away
14320 NE 20th St	Bellevue, WA 98007	Bellevue	98007	18:00 PM	0.59 miles away
14339 NE 20th St	Bellevue, WA 98007	Bellevue	98007	22:00 PM	0.61 miles away
2241 148th Ave NE	Bellevue, WA 98007	Bellevue	98007	22:00 PM	0.78 miles away
2285 140th Ave NE	Bellevue, WA 98005	Bellevue	98005	18:00 PM	0.84 miles away

To maximize the usefulness of this store locator, we include a responsive layout that adjusts when a user's screen width is smaller than 700 pixels wide. A responsive layout makes it easy to use the store locator on a small screen, like on a mobile device. Here's a wireframe of the small-screen layout:



Find a store

14320 NE 20th St
Bellevue, WA 98007
Open until 18:00 PM
0.59 miles away
[\(206\) 555-0128](tel:(206)555-0128)

1299 156th Ave NE
Bellevue, WA 98007
Open until 18:00 PM
0.51 miles away

14320 NE 20th St
Bellevue, WA 98007
Open until 18:00 PM
0.59 miles away

14339 NE 20th St

Create the store location dataset

This section describes how to create a dataset of the stores that you want to display on the map. The dataset for the Contoso Coffee locator is created inside an Excel workbook. The dataset contains 10,213 Contoso Coffee coffee shop locations spread across nine countries or regions: the United States, Canada, the United Kingdom, France, Germany, Italy, the Netherlands, Denmark, and Spain. Here's a screenshot of what the data looks like:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	AddressLine	City	Municipality	AdminDivision	Country	PostCode	Latitude	Longitude	Phone	IsWiFiHotSpot	IsWheelchairAccessible	Opens	Closes
2	12669 S Dixie Hwy	Miami		FL	US	33156	25.65385	-80.32947	(239) 555-0168	False	True	600	2300
3	350 Ocean Dr	Miami Beach		FL	US	33139	25.77258	-80.13225	(239) 555-0181	False	True	600	2300
4	6655 Ashcroft Dr	Houston		TX	US	77081	29.70894	-95.49112	(210) 555-0117	False	True	600	2300
5	2941 N Clark St	Chicago		IL	US	60657	41.93603	-87.64775	(217) 555-0138	True	True	900	2400
6	3 North St	Windsor Locks		CT	US	06096	41.93644	-72.63067	(203) 555-0134	False	False	700	1800
7	3144 N Narragansett Ave	Chicago		IL	US	60634	41.93676	-87.78566	(217) 555-0164	False	True	700	1800
8	1094 Bay St	Taunton		MA	US	02780	41.93703	-71.11111	(339) 555-0190	False	True	800	2200
9	28 Pleasant Rd	Enfield		CT	US	06082	41.93742	-72.60954	(203) 555-0101	False	True	700	1800
10	169 Pine St	Attleboro		MA	US	02703	41.9375	-71.28221	(339) 555-0166	False	True	800	2200
11	517 W Lake St	Addison		IL	US	60101	41.93757	-88.00082	(217) 555-0132	True	True	700	1800
12	503 W Lake St	Addison		IL	US	60101	41.93768	-88.00106	(217) 555-0192	False	False	700	1800
13	613 W Lake St	Addison		IL	US	60101	41.93898	-88.00413	(217) 555-0105	False	False	700	1800
14	332 W Army Trail Rd	Bloomingdale		IL	US	60108	41.93914	-88.10544	(217) 555-0112	False	True	700	1800
15	2441 University Blvd	Houston		TX	US	77005	29.71493	-95.41568	(210) 555-0131	False	True	800	2200
16	3215 N Sheffield Ave	Chicago		IL	US	60657	41.94124	-87.65433	(217) 555-0191	False	True	800	2200
17	3426 N Harlem Ave	Chicago		IL	US	60634	41.94286	-87.80675	(217) 555-0116	False	True	800	2200

To view the full dataset, [download the Excel workbook here](#).

Looking at the screenshot of the data, we can make the following observations:

- Location information is stored by using the **AddressLine**, **City**, **Municipality** (county), **AdminDivision** (state/province), **PostCode** (postal code), and **Country** columns.
- The **Latitude** and **Longitude** columns contain the coordinates for each Contoso Coffee location. If you don't have coordinates information, you can use the Search services in Azure Maps to determine the location coordinates.
- Some other columns contain metadata that's related to the coffee shops: a phone number, Boolean columns, and store opening and closing times in 24-hour format. The Boolean columns are for Wi-Fi and wheelchair accessibility.

accessibility. You can create your own columns that contain metadata that's more relevant to your location data.

NOTE

Azure Maps renders data in the spherical Mercator projection "EPSG:3857" but reads data in "EPSG:4326" that use the WGS84 datum.

Load the store location dataset

The Contoso Coffee shop locator dataset is small, so we'll convert the Excel worksheet into a tab-delimited text file. This file can then be downloaded by the browser when the application loads.

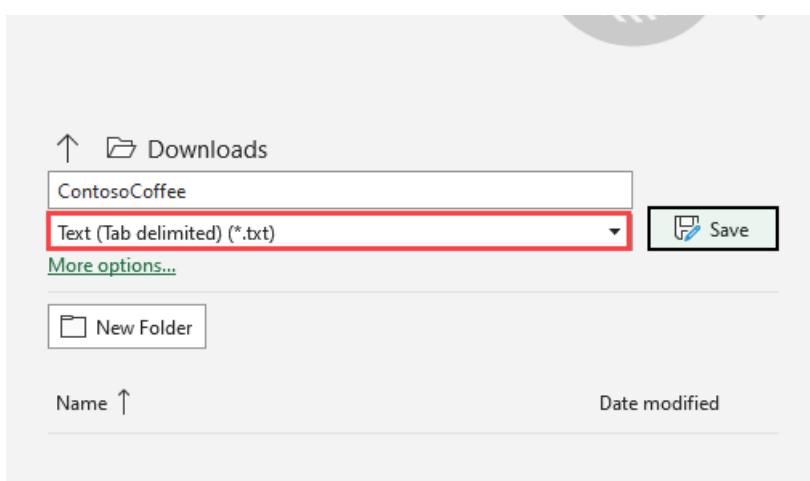
TIP

If your dataset is too large for client download, or is updated frequently, you might consider storing your dataset in a database. After your data is loaded into a database, you can then set up a web service that accepts queries for the data, and then sends the results to the user's browser.

Convert data to tab-delimited text file

To convert the Contoso Coffee shop location data from an Excel workbook into a flat text file:

1. [Download the Excel workbook](#).
2. Save the workbook to your hard drive.
3. Load the Excel app.
4. Open the downloaded workbook.
5. Select **Save As**.
6. In the **Save as type** drop-down list, select **Text (Tab delimited) (*.txt)**.
7. Name the file *ContosoCoffee*.



If you open the text file in Notepad, it looks similar to the following text:

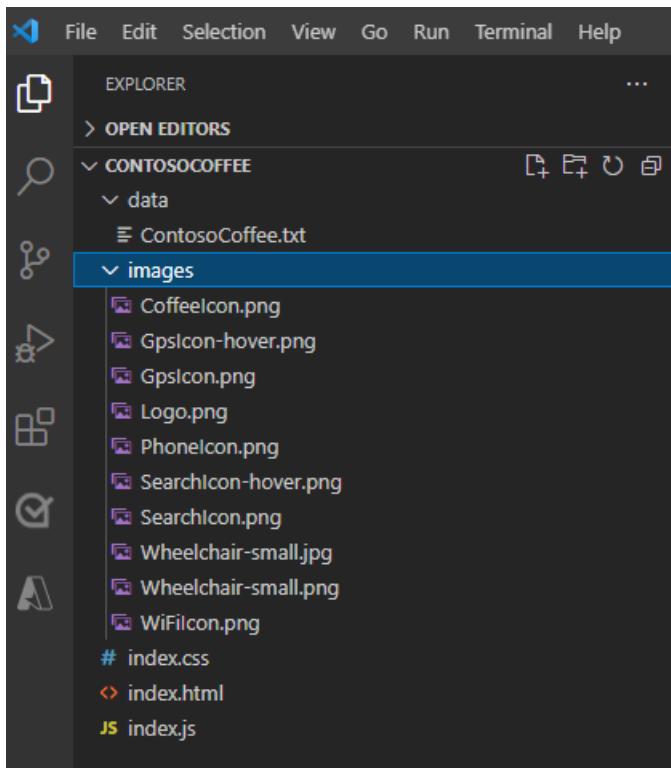
ContosoCoffee.txt - Notepad

AddressLine	City	Municipality	AdminDivision	Country	PostCode	Latitude	Longitude
935 N Krome Ave Homestead			FL	US	33034 25.45709	-80.47742	(239) 555-
12669 S Dixie Hwy	Miami		FL	US	33156 25.65385	-80.32947	(239) 555-
350 Ocean Dr	Miami Beach		FL	US	33139 25.77258	-80.13225	(239) 555-
6655 Ashcroft Dr	Houston		TX	US	77081 29.70894	-95.49112	(210) 555-
622 George Washington Hwy	Lincoln		RI	US	02865 41.93546	-71.4808	(4)
1615 W Wellington Ave	Chicago		IL	US	60657 41.93589	-87.66917	(217) 555-
2941 N Clark St	Chicago		IL	US	60657 41.93603	-87.64775	(217) 555-0138 Tr
3 North St	Windsor Locks		CT	US	06096 41.93644	-72.63067	(203) 555-
305 Army Trail Rd	Glendale Heights		IL	US	60634 41.93676	-87.78566	(217) 555-
3144 N Narragansett Ave	Chicago		IL	US	60657 41.93681	-87.65394	(217) 555-
3002 N Sheffield Ave	Chicago		IL	US	06082 41.93742	-72.60954	(203) 555-0101 Fe
1094 Bay St	Taunton	MA	US	02780 41.93703	-71.11111	(339) 555-0190 Fe	
28 Pleasant Rd	Enfield	CT	US	60634 41.93676	-87.78566	(217) 555-	
7126 Clarewood Dr	Houston		TX	US	77036 29.70961	-95.51303	(210) 555-
2150 Bloomingdale Rd	Glendale Heights		IL	US	60139 41.93747	-88.0801	
169 Pine St	Attleboro	MA	US	02703 41.9375	-71.28221	(339) 555-0166 Fe	

Set up the project

1. Open the Visual Studio Code app.
2. Select **File**, and then select **Open Workspace....**
3. Create a new folder and name it "ContosoCoffee".
4. Select **CONTOSOCOFFEE** in the explorer.
5. Create the following three files that define the layout, style, and logic for the application:
 - *index.html*
 - *index.css*
 - *index.js*
6. Create a folder named *data*.
7. Add *ContosoCoffee.txt* to the *data* folder.
8. Create another folder named *images*.
9. If you haven't already, [download these 10 images](#).
10. Add the downloaded images to the *images* folder.

Your workspace folder should now look like the following screenshot:



Create the HTML

To create the HTML:

1. Add the following `meta` tags to the `head` of `index.html`:

```
<meta charset="utf-8">
<meta http-equiv="x-ua-compatible" content="IE=Edge">
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

2. Add references to the Azure Maps web control JavaScript and CSS files:

```
<link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css"
type="text/css">
<script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>
```

3. Add a reference to the Azure Maps Services module. The module is a JavaScript library that wraps the Azure Maps REST services and makes them easy to use in JavaScript. The module is useful for powering search functionality.

```
<script src="https://atlas.microsoft.com/sdk/javascript/service/2/atlas-service.min.js"></script>
```

4. Add references to `index.js` and `index.css`.

```
<link rel="stylesheet" href="index.css" type="text/css">
<script src="index.js"></script>
```

5. In the body of the document, add a `header` tag. Inside the `header` tag, add the logo and company name.

```
<header>
  
  <span>Contoso Coffee</span>
</header>
```

6. Add a `main` tag and create a search panel that has a text box and search button. Also, add `div` references for the map, the list panel, and the My Location GPS button.

```
<main>
  <div class="searchPanel">
    <div>
      <input id="searchTbx" type="search" placeholder="Find a store" />
      <button id="searchBtn" title="Search"></button>
    </div>
  </div>
  <div id="listPanel"></div>
  <div id="myMap"></div>
  <button id="myLocationBtn" title="My Location"></button>
</main>
```

After you finish, `index.htm` should look like [this example index.html file](#).

Define the CSS Styles

The next step is to define the CSS styles. CSS styles define how the application components are laid out and the application's appearance.

1. Open `index.css`.
2. Add the following css code:

NOTE

The `@media` style defines alternate style options to use when the screen width is smaller than 700 pixels.

```
html, body {
  padding: 0;
  margin: 0;
  font-family: Gotham, Helvetica, sans-serif;
  overflow-x: hidden;
}

header {
  width: calc(100vw - 10px);
  height: 30px;
  padding: 15px 0 20px 20px;
  font-size: 25px;
  font-style: italic;
  font-family: "Comic Sans MS", cursive, sans-serif;
  line-height: 30px;
  font-weight: bold;
  color: white;
  background-color: #007faa;
}

header span {
  vertical-align: middle;
}

header img {
```

```
    height: 30px;
    vertical-align: middle;
}

.searchPanel {
    position: relative;
    width: 350px;
}

.searchPanel div {
    padding: 20px;
}

.searchPanel input {
    width: calc(100% - 50px);
    font-size: 16px;
    border: 0;
    border-bottom: 1px solid #ccc;
}

#listPanel {
    position: absolute;
    top: 135px;
    left: 0px;
    width: 350px;
    height: calc(100vh - 135px);
    overflow-y: auto;
}

#myMap {
    position: absolute;
    top: 65px;
    left: 350px;
    width: calc(100vw - 350px);
    height: calc(100vh - 65px);
}

.statusMessage {
    margin: 10px;
}

#myLocationBtn, #searchBtn {
    margin: 0;
    padding: 0;
    border: none;
    border-collapse: collapse;
    width: 32px;
    height: 32px;
    text-align: center;
    cursor: pointer;
    line-height: 32px;
    background-repeat: no-repeat;
    background-size: 20px;
    background-position: center center;
    z-index: 200;
}

#myLocationBtn {
    position: absolute;
    top: 150px;
    right: 10px;
    box-shadow: 0px 0px 4px rgba(0,0,0,0.16);
    background-color: white;
    background-image: url("images/GpsIcon.png");
}

#myLocationBtn:hover {
    background-image: url("images/GpsIcon-hover.png");
}
```

```
#searchBtn {  
    background-color: transparent;  
    background-image: url("images/SearchIcon.png");  
}  
  
.listItem {  
    height: 50px;  
    padding: 20px;  
    font-size: 14px;  
}  
  
.listItem:hover {  
    cursor: pointer;  
    background-color: #f1f1f1;  
}  
  
.listItem-title {  
    color: #007faa;  
    font-weight: bold;  
}  
  
.storePopup {  
    min-width: 150px;  
}  
  
.storePopup .popupTitle {  
    border-top-left-radius: 4px;  
    border-top-right-radius: 4px;  
    padding: 8px;  
    height: 30px;  
    background-color: #007faa;  
    color: white;  
    font-weight: bold;  
}  
  
.storePopup .popupSubTitle {  
    font-size: 10px;  
    line-height: 12px;  
}  
  
.storePopup .popupContent {  
    font-size: 11px;  
    line-height: 18px;  
    padding: 8px;  
}  
  
.storePopup img {  
    vertical-align:middle;  
    height: 12px;  
    margin-right: 5px;  
}  
  
/* Adjust the layout of the page when the screen width is fewer than 700 pixels. */  
@media screen and (max-width: 700px) {  
    .searchPanel {  
        width: 100vw;  
    }  
  
    #listPanel {  
        top: 385px;  
        width: 100%;  
        height: calc(100vh - 385px);  
    }  
}
```

```

#myMap {
    width: 100vw;
    height: 250px;
    top: 135px;
    left: 0px;
}

#myLocationBtn {
    top: 220px;
}
}

.mapCenterIcon {
    display: block;
    width: 10px;
    height: 10px;
    border-radius: 50%;
    background: orange;
    border: 2px solid white;
    cursor: pointer;
    box-shadow: 0 0 0 rgba(0, 204, 255, 0.4);
    animation: pulse 3s infinite;
}

@keyframes pulse {
    0% {
        box-shadow: 0 0 0 0 rgba(0, 204, 255, 0.4);
    }

    70% {
        box-shadow: 0 0 0 50px rgba(0, 204, 255, 0);
    }

    100% {
        box-shadow: 0 0 0 0 rgba(0, 204, 255, 0);
    }
}

```

Run the application. You'll see the header, search box, and search button. However, the map isn't visible because it hasn't been loaded yet. If you try to do a search, nothing happens. We need to set up the JavaScript logic, which is described in the next section. This logic accesses all the functionality of the store locator.

Add JavaScript code

The JavaScript code in the Contoso Coffee shop locator app enables the following processes:

1. Adds an **event listener** called `ready` to wait until the page has completed its loading process. When the page loading is complete, the event handler creates more event listeners to monitor the loading of the map, and give functionality to the **search** and **My location** buttons.
2. When the user selects the search button, or types a location in the search box then presses enter, a fuzzy search against the user's query is started. The code passes in an array of country/region ISO 2 values to the `countrySet` option to limit the search results to those countries/regions. Limiting the countries/regions to search helps increase the accuracy of the results that are returned.
3. Once the search is finished, the first location result is used as the center focus of the map camera. When the user selects the My Location button, the code retrieves the user's location using the *HTML5 Geolocation API* that's built into the browser. After retrieving the location, the code centers the map over the user's location.

To add the JavaScript:

1. Open *index.js*.
2. Add global options to make settings easier to update. Define the variables for the map, pop up window, data source, icon layer, and HTML marker. Set the HTML marker to indicate the center of a search area. And, define an instance of the Azure Maps search service client.

```
//The maximum zoom level to cluster data point data on the map.
var maxClusterZoomLevel = 11;

//The URL to the store location data.
var storeLocationDataUrl = 'data/ContosoCoffee.txt';

//The URL to the icon image.
var iconImageUrl = 'images/CoffeeIcon.png';
var map, popup, datasource, iconLayer, centerMarker, searchURL;
```

3. Add the following initialization code. Make sure to replace <Your Azure Maps Key> with your primary subscription key.

TIP

When you use pop-up windows, it's best to create a single `Popup` instance and reuse the instance by updating its content and position. For every `Popup` instance you add to your code, multiple DOM elements are added to the page. The more DOM elements there are on a page, the more things the browser has to keep track of. If there are too many items, the browser might become slow.

```
function initialize() {
    //Initialize a map instance.
    map = new atlas.Map('myMap', {
        center: [-90, 40],
        zoom: 2,

        //Add your Azure Maps primary subscription key to the map SDK.
        authOptions: {
            authType: 'subscriptionKey',
            subscriptionKey: '<Your Azure Maps Key>'
        }
    });

    //Create a pop-up window, but leave it closed so we can update it and display it later.
    popup = new atlas.Popup();

    //Use SubscriptionKeyCredential with a subscription key
    const subscriptionKeyCredential = new
atlas.service.SubscriptionKeyCredential(atlas.getSubscriptionKey());

    //Use subscriptionKeyCredential to create a pipeline
    const pipeline = atlas.serviceMapsURL.newPipeline(subscriptionKeyCredential, {
        retryOptions: { maxTries: 4 } // Retry options
    });

    //Create an instance of the SearchURL client.
    searchURL = new atlas.service.SearchURL(pipeline);

    //If the user selects the search button, geocode the value the user passed in.
    document.getElementById('searchBtn').onclick = performSearch;

    //If the user presses Enter in the search box, perform a search.
    document.getElementById('searchTbx').onkeyup = function(e) {
        if (e.keyCode === 13) {
            performSearch();
        }
    };
}
```

```

        }

    //If the user selects the My Location button, use the Geolocation API (Preview) to get the user's
    //location. Center and zoom the map on that location.
    document.getElementById('myLocationBtn').onclick = setMapToUserLocation;

    //Wait until the map resources are ready.
    map.events.add('ready', function() {

        //Add your post-map load functionality.

    });
}

//Create an array of country/region ISO 2 values to limit searches to.
var countrySet = ['US', 'CA', 'GB', 'FR','DE','IT','ES','NL','DK'];

function performSearch() {
    var query = document.getElementById('searchTbx').value;

    //Perform a fuzzy search on the users query.
    searchURL.searchFuzzy(atlas.service.Aborts.timeout(3000), query, {
        //Pass in the array of country/region ISO2 for which we want to limit the search to.
        countrySet: countrySet
    }).then(results => {
        //Parse the response into GeoJSON so that the map can understand.
        var data = results.geojson.getFeatures();

        if (data.features.length > 0) {
            //Set the camera to the bounds of the results.
            map.setCamera({
                bounds: data.features[0].bbox,
                padding: 40
            });
        } else {
            document.getElementById('listPanel').innerHTML = '<div class="statusMessage">Unable to
find the location you searched for.</div>';
        }
    });
}

function setMapToUserLocation() {
    //Request the user's location.
    navigator.geolocation.getCurrentPosition(function(position) {
        //Convert the Geolocation API (Preview) position to a longitude and latitude position value
        //that the map can interpret and center the map over it.
        map.setCamera({
            center: [position.coords.longitude, position.coords.latitude],
            zoom: maxClusterZoomLevel + 1
        });
    }, function(error) {
        //If an error occurs when the API tries to access the user's position information, display an
        //error message.
        switch (error.code) {
            case error.PERMISSION_DENIED:
                alert('User denied the request for geolocation.');
                break;
            case error.POSITION_UNAVAILABLE:
                alert('Position information is unavailable.');
                break;
            case error.TIMEOUT:
                alert('The request to get user position timed out.');
                break;
            case error.UNKNOWN_ERROR:
                alert('An unknown error occurred.');
                break;
        }
    });
}

```

```

}

//Initialize the application when the page is loaded.
window.onload = initialize;

```

4. In the map's `ready` event listener, add a zoom control and an HTML marker to display the center of a search area.

```

//Add a zoom control to the map.
map.controls.add(new atlas.control.ZoomControl(), {
    position: 'top-right'
});

//Add an HTML marker to the map to indicate the center to use for searching.
centerMarker = new atlas.HtmlMarker({
    htmlContent: '<div class="mapCenterIcon"></div>',
    position: map.getCamera().center
});

map.markers.add(centerMarker);

```

5. In the map's `ready` event listener, add a data source. Then, make a call to load and parse the dataset.

Enable clustering on the data source. Clustering on the data source groups overlapping points together in a cluster. As the user zooms in, the clusters separate into individual points. This behavior provides a better user experience and improves performance.

```

//Create a data source, add it to the map, and then enable clustering.
datasource = new atlas.source.DataSource(null, {
    cluster: true,
    clusterMaxZoom: maxClusterZoomLevel - 1
});

map.sources.add(datasource);

//Load all the store data now that the data source is defined.
loadStoreData();

```

6. After the dataset loads in the map's `ready` event listener, define a set of layers to render the data. A bubble layer renders clustered data points. A symbol layer renders the number of points in each cluster above the bubble layer. A second symbol layer renders a custom icon for individual locations on the map.

Add `mouseover` and `mouseout` events to the bubble and icon layers to change the mouse cursor when the user hovers over a cluster or icon on the map. Add a `click` event to the cluster bubble layer. This `click` event zooms in the map two levels and centers the map over a cluster when the user selects any cluster. Add a `click` event to the icon layer. This `click` event displays a pop-up window that shows the details of a coffee shop when a user selects an individual location icon. Add an event to the map to monitor when the map is finished moving. When this event fires, update the items in the list panel.

```

//Create a bubble layer to render clustered data points.
var clusterBubbleLayer = new atlas.layer.BubbleLayer(datasource, null, {
    radius: 12,
    color: '#007faa',
    strokeColor: 'white',
    strokeWidth: 2,
    filter: ['has', 'point_count'] //Only render data points that have a point_count property;
    clusters have this property.
});

//Create a symbol layer to render the count of locations in a cluster.
var clusterSymbolLayer = new atlas.layer.SymbolLayer(datasource, null, {

```

```

var clusterLabelLayer = new atlas.layer.SymbolLayer(datasource, null, {
    iconOptions: {
        image: 'none' //Hide the icon image.
    },
    textOptions: {
        textField: ['get', 'point_count_abbreviated'],
        size: 12,
        font: ['StandardFont-Bold'],
        offset: [0, 0.4],
        color: 'white'
    }
});

map.layers.add([clusterBubbleLayer, clusterLabelLayer]);

//Load a custom image icon into the map resources.
map.imageSprite.add('myCustomIcon', iconImageUrl).then(function() {

    //Create a layer to render a coffee cup symbol above each bubble for an individual location.
    iconLayer = new atlas.layer.SymbolLayer(datasource, null, {
        iconOptions: {
            //Pass in the ID of the custom icon that was loaded into the map resources.
            image: 'myCustomIcon',
            //Optionally, scale the size of the icon.
            font: ['SegoeUi-Bold'],
            //Anchor the center of the icon image to the coordinate.
            anchor: 'center',
            //Allow the icons to overlap.
            allowOverlap: true
        },
        filter: ['!', ['has', 'point_count']] //Filter out clustered points from this layer.
    });

    map.layers.add(iconLayer);

    //When the mouse is over the cluster and icon layers, change the cursor to a pointer.
    map.events.add('mouseover', [clusterBubbleLayer, iconLayer], function() {
        map.getCanvasContainer().style.cursor = 'pointer';
    });

    //When the mouse leaves the item on the cluster and icon layers, change the cursor back to the default (grab).
    map.events.add('mouseout', [clusterBubbleLayer, iconLayer], function() {
        map.getCanvasContainer().style.cursor = 'grab';
    });

    //Add a click event to the cluster layer. When the user selects a cluster, zoom into it by two levels.
    map.events.add('click', clusterBubbleLayer, function(e) {
        map.setCamera({
            center: e.position,
            zoom: map.getCamera().zoom + 2
        });
    });

    //Add a click event to the icon layer and show the shape that was selected.
    map.events.add('click', iconLayer, function(e) {
        showPopup(e.shapes[0]);
    });

    //Add an event to monitor when the map is finished rendering the map after it has moved.
    map.events.add('render', function() {
        //Update the data in the list.
        updateListItems();
    });
});

```

```
});
```

7. When the coffee shop dataset is loaded, it must first be downloaded. Then, the text file must be split into lines. The first line contains the header information. To make the code easier to follow, we parse the header into an object, which we can then use to look up the cell index of each property. After the first line, loop through the remaining lines and create a point feature. Add the point feature to the data source. Finally, update the list panel.

```

function loadStoreData() {

    //Download the store location data.
    fetch(storeLocationDataURL)
        .then(response => response.text())
        .then(function(text) {

            //Parse the tab-delimited file data into GeoJSON features.
            var features = [];

            //Split the lines of the file.
            var lines = text.split('\n');

            //Grab the header row.
            var row = lines[0].split('\t');

            //Parse the header row and index each column to make the code for parsing each row easier to
            follow.
            var header = {};
            var numColumns = row.length;
            for (var i = 0; i < row.length; i++) {
                header[row[i]] = i;
            }

            //Skip the header row and then parse each row into a GeoJSON feature.
            for (var i = 1; i < lines.length; i++) {
                row = lines[i].split('\t');

                //Ensure that the row has the correct number of columns.
                if (row.length >= numColumns) {

                    features.push(new atlas.data.Feature(new
atlas.data.Point([parseFloat(row[header['Longitude']]), parseFloat(row[header['Latitude']])]), {
                        AddressLine: row[header['AddressLine']],
                        City: row[header['City']],
                        Municipality: row[header['Municipality']],
                        AdminDivision: row[header['AdminDivision']],
                        Country: row[header['Country']],
                        PostCode: row[header['PostCode']],
                        Phone: row[header['Phone']],
                        StoreType: row[header['StoreType']],
                        IsWiFiHotSpot: (row[header['IsWiFiHotSpot']].toLowerCase() === 'true') ? true :
false,
                        IsWheelchairAccessible: (row[header['IsWheelchairAccessible']].toLowerCase() ===
'true') ? true : false,
                        Opens: parseInt(row[header['Opens']]),
                        Closes: parseInt(row[header['Closes']])
                    }));
                }
            }

            //Add the features to the data source.
            datasource.add(new atlas.data.FeatureCollection(features));

            //Initially, update the list items.
            updateListItems();
        });
    }
}

```

- When the list panel is updated, the distance is calculated. This distance is from the center of the map to all point features in the current map view. The features are then sorted by distance. HTML is generated to display each location in the list panel.

```

var listItemTemplate = '<div class="listItem" onclick="itemSelected(\'{id}\')"><div class="listItem-
title">{title}</div>{city}<br />Open until {closes}<br />{distance} miles away</div>';

```

```

function updateListItems() {
    //Hide the center marker.
    centerMarker.setOptions({
        visible: false
    });

    //Get the current camera and view information for the map.
    var camera = map.getCamera();
    var listPanel = document.getElementById('listPanel');

    //Check to see whether the user is zoomed out a substantial distance. If they are, tell the user
    to zoom in and to perform a search or select the My Location button.
    if (camera.zoom < maxClusterZoomLevel) {
        //Close the pop-up window; clusters might be displayed on the map.
        popup.close();
        listPanel.innerHTML = '<div class="statusMessage">Search for a location, zoom the map, or
        select the My Location button to see individual locations.</div>';
    } else {
        //Update the location of the centerMarker property.
        centerMarker.setOptions({
            position: camera.center,
            visible: true
        });
    }

    //List the ten closest locations in the side panel.
    var html = [], properties;

    /*
    Generating HTML for each item that looks like this:
    <div class="listItem" onclick="itemSelected('id')">
        <div class="listItem-title">1 Microsoft Way</div>
        Redmond, WA 98052<br />
        Open until 9:00 PM<br />
        0.7 miles away
    </div>
    */

    //Get all the shapes that have been rendered in the bubble layer.
    var data = map.layers.getRenderedShapes(map.getCamera().bounds, [iconLayer]);

    //Create an index of the distances of each shape.
    var distances = {};

    data.forEach(function (shape) {
        if (shape instanceof atlas.Shape) {

            //Calculate the distance from the center of the map to each shape and store in the
            index. Round to 2 decimals.
            distances[shape.getId()] = Math.round(atlas.math.getDistanceTo(camera.center,
            shape.getCoordinates(), 'miles') * 100) / 100;
        }
    });

    //Sort the data by distance.
    data.sort(function (x, y) {
        return distances[x.getId()] - distances[y.getId()];
    });

    data.forEach(function(shape) {
        properties = shape.getProperties();
        html.push('<div class="listItem" onclick="itemSelected(\'', shape.getId(), '\')"><div
        class="listItem-title">',
            properties['AddressLine'],
            '</div>',
            //Get a formatted addressLine2 value that consists of City, Municipality, AdminDivision,
            and PostCode.
            getAddressLine2(properties),
            '<br />',

```

```

        //Convert the closing time to a format that is easier to read.
        getOpenTillTime(properties),
        '<br />',

        //Get the distance of the shape.
        distances[shape.getId()],
        ' miles away</div>');
    });

    listPanel.innerHTML = html.join('');

    //Scroll to the top of the list panel in case the user has scrolled down.
    listPanel.scrollTop = 0;
}
}

//This converts a time that's in a 24-hour format to an AM/PM time or noon/midnight string.
function getOpenTillTime(properties) {
    var time = properties['Closes'];
    var t = time / 100;
    var sTime;

    if (time === 1200) {
        sTime = 'noon';
    } else if (time === 0 || time === 2400) {
        sTime = 'midnight';
    } else {
        sTime = Math.round(t) + ':';

        //Get the minutes.
        t = (t - Math.round(t)) * 100;

        if (t === 0) {
            sTime += '00';
        } else if (t < 10) {
            sTime += '0' + t;
        } else {
            sTime += Math.round(t);
        }

        if (time < 1200) {
            sTime += ' AM';
        } else {
            sTime += ' PM';
        }
    }

    return 'Open until ' + sTime;
}

//Create an addressLine2 string that contains City, Municipality, AdminDivision, and PostCode.
function getAddressLine2(properties) {
    var html = [properties['City']];

    if (properties['Municipality']) {
        html.push(', ', properties['Municipality']);
    }

    if (properties['AdminDivision']) {
        html.push(', ', properties['AdminDivision']);
    }

    if (properties['PostCode']) {
        html.push(' ', properties['PostCode']);
    }

    return html.join('');
}

```

9. When the user selects an item in the list panel, the shape to which the item is related is retrieved from the data source. A pop-up window is generated that's based on the property information stored in the shape. The map centers over the shape. If the map is fewer than 700 pixels wide, the map view is offset so the pop-up window is visible.

```

//When a user selects a result in the side panel, look up the shape by its ID value and display the
//pop-up window.
function itemSelected(id) {
    //Get the shape from the data source by using its ID.
    var shape = datasource.getShapeById(id);
    showPopup(shape);

    //Center the map over the shape on the map.
    var center = shape.getCoordinates();
    var offset;

    //If the map is fewer than 700 pixels wide, then the layout is set for small screens.
    if (map.getCanvas().width < 700) {
        //When the map is small, offset the center of the map relative to the shape so that there is
        room for the popup to appear.
        offset = [0, -80];
    }

    map.setCamera({
        center: center,
        centerOffset: offset
    });
}

function showPopup(shape) {
    var properties = shape.getProperties();

    /* Generating HTML for the pop-up window that looks like this:

        <div class="storePopup">
            <div class="popupTitle">
                3159 Tongass Avenue
                <div class="popupSubTitle">Ketchikan, AK 99901</div>
            </div>
            <div class="popupContent">
                Open until 22:00 PM<br/>
                
                <a href="tel:1-800-XXX-XXXX">1-800-XXX-XXXX</a>
                <br>Amenities:
                
                
            </div>
        </div>
    */

    //Calculate the distance from the center of the map to the shape in miles, round to 2 decimals.
    var distance = Math.round(atlas.math.getDistanceTo(map.getCamera().center,
        shape.getCoordinates(), 'miles') * 100)/100;

    var html = ['<div class="storePopup">'];
    html.push('<div class="popupTitle">',
        properties['AddressLine'],
        '<div class="popupSubTitle">',
        getAddressLine2(properties),
        '</div></div><div class="popupContent">',

        //Convert the closing time to a format that's easier to read.
        getOpenTillTime(properties),

        //Add the distance information.
    ]
}

```

```

'<br/>', distance,
'miles away',
'<br /><a href="tel:',
properties['Phone'],
'">',
properties['Phone'],
'</a>'
);

if (properties['IsWiFiHotSpot'] || properties['IsWheelchairAccessible']) {
    html.push('<br/>Amenities: ');

    if (properties['IsWiFiHotSpot']) {
        html.push('');
    }

    if (properties['IsWheelchairAccessible']) {
        html.push('');
    }
}

html.push('</div></div>');

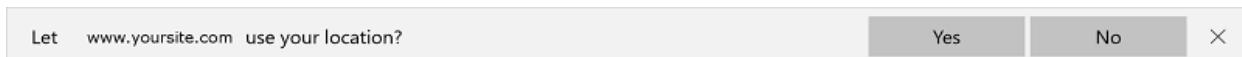
//Update the content and position of the pop-up window for the specified shape information.
popup.setOptions({
    //Create a table from the properties in the feature.
    content: html.join(),
    position: shape.getCoordinates()
});

//Open the pop-up window.
popup.open(map);
}

```

Now, you have a fully functional store locator. In a web browser, open the *index.html* file for the store locator. When the clusters appear on the map, you can search for a location by using the search box, by selecting the My Location button, by selecting a cluster, or by zooming in on the map to see individual locations.

The first time a user selects the My Location button, the browser displays a security warning that asks for permission to access the user's location. If the user agrees to share their location, the map zooms in on the user's location, and nearby coffee shops are shown.



When you zoom in close enough in an area that has coffee shop locations, the clusters separate into individual locations. Select one of the icons on the map or select an item in the side panel to see a pop-up window. The pop-up shows information for the selected location.

Contoso Coffee

98052

8042 161st Ave NE
Redmond, WA 98052
Open until 20:00 PM
0.42 miles away

16564 Cleveland St
Redmond, WA 98052
Open until 22:00 PM
0.61 miles away

17120 Redmond Way
Redmond, WA 98052
Open until 23:30 PM
0.9 miles away

6501 132nd Ave NE
Kirkland, WA 98033
Open until 18:00 PM
2.39 miles away

8514 122nd Ave NE
Kirkland, WA 98033
Open until 22:00 PM
2.67 miles away

8506 122nd Ave NE
Kirkland, WA 98033
Open until 23:00 PM
2.67 miles away

16564 Cleveland St
Redmond, WA 98052
Open until 22:00 PM
0.61 miles away
Call: (206) 555-0195
Amenities: ⚡

Microsoft
© 1992 - 2018 TomTom

If you resize the browser window to fewer than 700 pixels wide or open the application on a mobile device, the layout changes to be better suited for smaller screens.

Contoso Coffee

Find a store

93 Pike St
Seattle, WA 98101
Open until midnight
0.07 miles away
Call: (206) 555-0114
Amenities: ⚡

2010 4th Ave
Seattle, WA 98121
Open until 20:00 PM
0.12 miles away

1909 3rd Ave
Seattle, WA 98101
Open until 22:00 PM
0.19 miles away

2219 4th Ave
Seattle, WA 98121
Open until 22:00 PM
0.10 miles away

Microsoft
© 1992 - 2018 TomTom

In this tutorial, you learned how to create a basic store locator by using Azure Maps. The store locator you create in this tutorial might have all the functionality you need. You can add features to your store locator or use more advance features for a more custom user experience:

- Enable [suggestions as you type](#) in the search box.
- Add [support for multiple languages](#).
- Allow the user to [filter locations along a route](#).
- Add the ability to [set filters](#).
- Add support to specify an initial search value by using a query string. When you include this option in your

store locator, users are then able to bookmark and share searches. It also provides an easy method for you to pass searches to this page from another page.

- Deploy your store locator as an [Azure App Service Web App](#).
- Store your data in a database and search for nearby locations. To learn more, see the [SQL Server spatial data types overview](#) and [Query spatial data for the nearest neighbor](#).

You can [view full source code here](#). [View the live sample](#) and learn more about the coverage and capabilities of Azure Maps by using [Zoom levels and tile grid](#). You can also [Use data-driven style expressions](#) to apply to your business logic.

Clean up resources

There are no resources that require cleanup.

Next steps

To see more code examples and an interactive coding experience:

[How to use the map control](#)

Tutorial: Load GeoJSON data into Azure Maps

Android SDK

3/5/2021 • 9 minutes to read • [Edit Online](#)

This tutorial guides you through the process of importing a GeoJSON file of location data into the Azure Maps Android SDK. In this tutorial, you learn how to:

- Add Azure Maps to an Android application.
- Create a data source and load in a GeoJSON file from a local file or the web.
- Display the data on the map.

Prerequisites

1. Complete the [Quickstart: Create an Android app](#). This tutorial will extend the code used in that quickstart.
2. Download the [Sample Points of Interest](#) GeoJSON file.

Import GeoJSON data from web or assets folder

Most GeoJSON files wrap all data within a `FeatureCollection`. With this in mind, if the GeoJSON files are loaded into the application as a string, they can be passed into the feature collection's static `fromJson` method, which will deserialize the string into a GeoJSON `FeatureCollection` object that can be added to the map.

The following steps show you how to import a GeoJSON file into the application and deserialize it as a GeoJSON `FeatureCollection` object.

1. Complete the [Quickstart: Create an Android app](#) as the following steps build on top of this application.
2. In the project panel of Android studio, right-click on the `app` folder and go to `New > Folder > Assets Folder`.
3. Drag and drop the [Sample Points of Interest](#) GeoJSON file into the assets folder.
4. Create a new file called `Utils.java` and add the following code to that file. This code provides a static method called `importData` that asynchronously imports a file from the `assets` folder of the application or from the web using a URL as a string and returns it back to the UI thread using a simple callback method.

```
//Modify the package name as needed to align with your application.
package com.example.myapplication;

import android.content.Context;
import android.os.Handler;
import android.os.Looper;
import android.webkit.URLUtil;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import javax.net.ssl.HttpsURLConnection;

public class Utils {

    interface SimpleCallback {
```

```

        void notify(String result);
    }

    /**
     * Imports data from a web url or asset file name and returns it to a callback.
     * @param urlOrFileName A web url or asset file name that points to data to load.
     * @param context The context of the app.
     * @param callback The callback function to return the data to.
     */
    public static void importData(String urlOrFileName, Context context, SimpleCallback callback){
        importData(urlOrFileName, context, callback, null);
    }

    /**
     * Imports data from a web url or asset file name and returns it to a callback.
     * @param urlOrFileName A web url or asset file name that points to data to load.
     * @param context The context of the app.
     * @param callback The callback function to return the data to.
     * @param error A callback function to return errors to.
     */
    public static void importData(String urlOrFileName, Context context, SimpleCallback callback,
SimpleCallback error){
        if(urlOrFileName != null && callback != null) {
            ExecutorService executor = Executors.newSingleThreadExecutor();
            Handler handler = new Handler(Looper.getMainLooper());

            executor.execute(() -> {
                String data = null;

                try {

                    if(URLUtil.isNetworkUrl(urlOrFileName)){
                        data = importFromWeb(urlOrFileName);
                    } else {
                        //Assume file is in assets folder.
                        data = importFromAssets(context, urlOrFileName);
                    }

                    final String result = data;

                    handler.post(() -> {
                        //Ensure the resulting data string is not null or empty.
                        if (result != null && !result.isEmpty()) {
                            callback.notify(result);
                        } else {
                            error.notify("No data imported.");
                        }
                    });
                } catch(Exception e) {
                    if(error != null){
                        error.notify(e.getMessage());
                    }
                }
            });
        }
    }

    /**
     * Imports data from an assets file as a string.
     * @param context The context of the app.
     * @param fileName The asset file name.
     * @return
     * @throws IOException
     */
    private static String importFromAssets(Context context, String fileName) throws IOException {
        InputStream stream = null;

        try {
            stream = context.getAssets().open(fileName);

```

```

        if(stream != null) {
            return readStreamAsString(stream);
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // Close Stream and disconnect HTTPS connection.
        if (stream != null) {
            stream.close();
        }
    }

    return null;
}

/**
 * Imports data from the web as a string.
 * @param url URL to the data.
 * @return
 * @throws IOException
 */
private static String importFromWeb(String url) throws IOException {
    InputStream stream = null;
    HttpsURLConnection connection = null;
    String result = null;

    try {
        connection = (HttpsURLConnection) new URL(url).openConnection();

        //For this use case, set HTTP method to GET.
        connection.setRequestMethod("GET");

        //Open communications link (network traffic occurs here).
        connection.connect();

        int responseCode = connection.getResponseCode();
        if (responseCode != HttpsURLConnection.HTTP_OK) {
            throw new IOException("HTTP error code: " + responseCode);
        }

        //Retrieve the response body as an InputStream.
        stream = connection.getInputStream();

        if (stream != null) {
            return readStreamAsString(stream);
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // Close Stream and disconnect HTTPS connection.
        if (stream != null) {
            stream.close();
        }
        if (connection != null) {
            connection.disconnect();
        }
    }

    return result;
}

/**
 * Reads an input stream as a string.
 * @param stream Stream to convert.
 * @return
 * @throws IOException
 */
private static String readStreamAsString(InputStream stream) throws IOException {

```

```

private static String reader(InputStream stream) throws IOException {
    //Convert the contents of an InputStream to a String.
    BufferedReader in = new BufferedReader(new InputStreamReader(stream, "UTF-8"));

    String inputLine;
    StringBuffer response = new StringBuffer();

    while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
    }

    in.close();

    return response.toString();
}
}

```

5. Go into the **MainActivity.java** file and add the following code inside the callback for the

`mapControl.onReady` event, this is located inside the `onCreate` method. This code uses the import utility to read in the **SamplePoiDataSet.json** file as a string, then deserializes it as a feature collection using the static `fromJson` method of the `FeatureCollection` class. This code also calculates the bounding box area for all the data in the feature collection and uses this to set the camera of the map to focus in on the data.

```

//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Import the GeoJSON data and add it to the data source.
Utils.importData("SamplePoiDataSet.json",
    this,
    (String result) -> {
        //Parse the data as a GeoJSON Feature Collection.
        FeatureCollection fc = FeatureCollection.fromJson(result);

        //Add the feature collection to the data source.
        source.add(fc);

        //Optionally, update the maps camera to focus in on the data.

        //Calculate the bounding box of all the data in the Feature Collection.
        BoundingBox bbox = MapMath.fromData(fc);

        //Update the maps camera so it is focused on the data.
        map.setCamera(
            bounds(bbox,

                //Padding added to account for pixel size of rendered points.
                padding(20)
            );
        );
    });

```

6. Using the code to load the GeoJSON data a data source, we now need to specify how that data should be displayed on the map. There are several different rendering layers for point data; [Bubble layer](#), [Symbol layer](#), and [Heat map layer](#) are the most commonly used layers. Add the following code to render the data in a bubble layer in the callback for the `mapControl.onReady` event after the code for importing the data.

```

//Create a layer and add it to the map.
BubbleLayer layer = new BubbleLayer(source);
map.layers.add(layer);

```

4. Create a new file called **Utils.kt** and add the following code to that file. This code provides a static

method called `importData` that asynchronously imports a file from the `assets` folder of the application or from the web using a URL as a string and returns it back to the UI thread using a simple callback method.

```

//Modify the package name as needed to align with your application.
package com.example.myapplication;

import android.content.Context
import android.os.Handler
import android.os.Looper
import android.webkit.URLUtil
import java.net.URL
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors

class Utils {
    companion object {

        /**
         * Imports data from a web url or asset file name and returns it to a callback.
         * @param urlOrFileName A web url or asset file name that points to data to load.
         * @param context The context of the app.
         * @param callback The callback function to return the data to.
         */
        fun importData(urlOrFileName: String?, context: Context, callback: (String?) -> Unit) {
            importData(urlOrFileName, context, callback, null)
        }

        /**
         * Imports data from a web url or asset file name and returns it to a callback.
         * @param urlOrFileName A web url or asset file name that points to data to load.
         * @param context The context of the app.
         * @param callback The callback function to return the data to.
         * @param error A callback function to return errors to.
         */
        public fun importData(urlOrFileName: String?, context: Context, callback: (String?) -> Unit,
error: ((String?) -> Unit)?) {
            if (urlOrFileName != null && callback != null) {
                val executor: ExecutorService = Executors.newSingleThreadExecutor()
                val handler = Handler(Looper.getMainLooper())
                executor.execute {
                    var data: String? = null

                    try {
                        data = if (URLUtil.isNetworkUrl(urlOrFileName)) {
                            URL(urlOrFileName).readText()
                        } else { //Assume file is in assets folder.
                            context.assets.open(urlOrFileName).bufferedReader().use{
                                it.readText()
                            }
                        }
                    }

                    handler.post {
                        //Ensure the resulting data string is not null or empty.
                        if (data != null && !data.isEmpty()) {
                            callback(data)
                        } else {
                            error!("No data imported.")
                        }
                    }
                }
            } catch (e: Exception) {
                error!(e.message)
            }
        }
    }
}

```

5. Go into the **MainActivity.kt** file and add the following code inside the callback for the

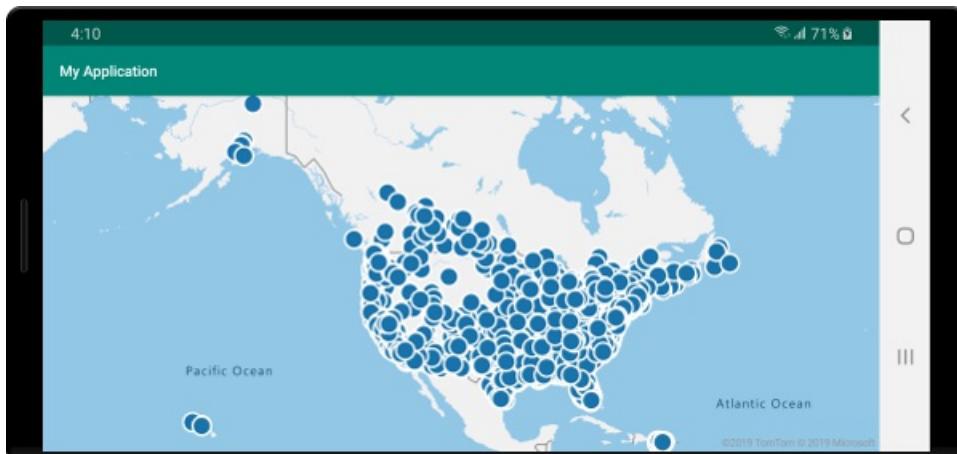
`mapControl.onReady` event, this is located inside the `onCreate` method. This code uses the import utility to read in the `SamplePoiDataSet.json` file as a string, then deserializes it as a feature collection using the static `fromJson` method of the `FeatureCollection` class. This code also calculates the bounding box area for all the data in the feature collection and uses this to set the camera of the map to focus in on the data.

```
//Create a data source and add it to the map.  
DataSource source = new DataSource();  
map.sources.add(source);  
  
//Import the GeoJSON data and add it to the data source.  
Utils.importData("SamplePoiDataSet.json", this) {  
    result: String? ->  
        //Parse the data as a GeoJSON Feature Collection.  
        val fc = FeatureCollection.fromJson(result!!)  
  
        //Add the feature collection to the data source.  
        source.add(fc)  
  
        //Optionally, update the maps camera to focus in on the data.  
  
        //Calculate the bounding box of all the data in the Feature Collection.  
        val bbox = MapMath.fromData(fc);  
  
        //Update the maps camera so it is focused on the data.  
        map.setCamera(  
            bounds(bbox),  
  
            //Padding added to account for pixel size of rendered points.  
            padding(20)  
        )  
}
```

- Using the code to load the GeoJSON data a data source, we now need to specify how that data should be displayed on the map. There are several different rendering layers for point data; [Bubble layer](#), [Symbol layer](#), and [Heat map layer](#) are the most commonly used layers. Add the following code to render the data in a bubble layer in the callback for the `mapControl.onReady` event after the code for importing the data.

```
//Create a layer and add it to the map.  
val layer = new BubbleLayer(source)  
map.layers.add(layer)
```

- Run the application. A map will be displayed focused over the USA, with circles overlaid for each location in the GeoJSON file.



Clean up resources

Take to following steps to clean up the resources from this tutorial:

1. Close Android Studio and delete the application you created.
2. If you tested the application on an external device, uninstall the application from that device.

Next steps

To see more code examples and an interactive coding experience:

[Use data-driven style expressions](#)

[Display feature information](#)

[Add a symbol layer](#)

[Add a line layer](#)

[Add a polygon layer](#)

Tutorial: Use Creator to create indoor maps

6/9/2021 • 12 minutes to read • [Edit Online](#)

This tutorial describes how to create indoor maps. In this tutorial, you'll learn how to:

- Upload your indoor map Drawing package.
- Convert your Drawing package into map data.
- Create a dataset from your map data.
- Create a tileset from the data in your dataset.
- Query the Azure Maps Web Feature Service (WFS) API to learn about your map features.
- Create a feature stateset by using your map features and the data in your dataset.
- Update your feature stateset.

Prerequisites

1. [Make an Azure Maps account](#).
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.
3. [Create a Creator resource](#).
4. Download the [Sample Drawing package](#).

This tutorial uses the [Postman](#) application, but you can use a different API development environment.

IMPORTANT

This tutorial uses the `us.atlas.microsoft.com` geographical URL. If your Creator service wasn't created in the United States, you must use a different geographical URL. For more information, see [Access to Creator Services](#). To view mappings of region to geographical location, [see Creator service geographic scope](#).

Upload a Drawing package

Use the [Data Upload API](#) to upload the Drawing package to Azure Maps resources.

The Data Upload API is a long running transaction that implements the pattern defined in [Creator Long-Running Operation API V2](#).

To upload the Drawing package:

1. In the Postman app, select **New**.
2. In the **Create New** window, select **Collection**.
3. Select **New** again.
4. In the **Create New** window, select **Request**.
5. Enter a **Request name** for the request, such as *POST Data Upload*.
6. Select the collection you previously created, and then select **Save**.
7. Select the **POST** HTTP method.
8. Enter the following URL to the [Data Upload API](#):

```
https://us.atlas.microsoft.com/mapData?api-version=2.0&dataFormat=dwgzippackage&subscription-key={Azure-Maps-Primary-Subscription-key}
```

IMPORTANT

For this request, and other requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key.

9. Select the **Headers** tab.
10. In the **KEY** field, select `Content-Type`.

11. In the **VALUE** field, select `application/octet-stream`.

The screenshot shows the Postman interface with the 'Headers' tab selected. There is one header entry displayed:

KEY	VALUE	DESCRIPTION
Content-Type	application/octet-stream	

12. Select the **Body** tab.
13. In the dropdown list, select **binary**.
14. Select **Select File**, and then select a Drawing package.

The screenshot shows the Postman interface with the 'Body' tab selected. The 'Content-Type' dropdown is set to 'binary'. Below it, there is a 'Select File' button.

15. Select **Send**.
16. In the response window, select the **Headers** tab.
17. Copy the value of the **Operation-Location** key, which is the `status URL`. We'll use the `status URL` to check the status of the Drawing package upload.

Cookies	Headers (9)	Test Results
KEY	VALUE	
Content-Type ⓘ	application/json	
x-ms-azuremaps-region ⓘ	West US 2	
X-Content-Type-Options ⓘ	nosniff	
Strict-Transport-Security ⓘ	max-age=31536000; includeSubDomains	
Operation-Location ⓘ	https://us.atlas.microsoft.com/mapData/operations/e781ebd4-0e59-4f7c-b4e8-118149d68a45?api-version=2.0	
X-Cache ⓘ	CONFIG_NOCACHE	
X-MSEdge-Ref ⓘ	Ref A: 6FF9BE162E0441099BA5369B69E1F1D4 Ref B: WSTEDGE1020 Ref C: 2021-05-19T18:17:27Z	
Date ⓘ	Wed, 19 May 2021 18:17:28 GMT	
Content-Length ⓘ	0	

Check the Drawing package upload status

To check the status of the drawing package and retrieve its unique ID (`udid`):

1. Select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name** for the request, such as *GET Data Upload Status*.
4. Select the collection you previously created, and then select **Save**.
5. Select the **GET** HTTP method.
6. Enter the `status URL` you copied in [Upload a Drawing package](#). The request should look like the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key):

```
https://us.atlas.microsoft.com/mapData/operations/<operationId>?api-version=2.0&subscription-key={Azure-Maps-Primary-Subscription-key}
```

7. Select **Send**.
8. In the response window, select the **Headers** tab.
9. Copy the value of the **Resource-Location** key, which is the `resource location URL`. The `resource location URL` contains the unique identifier (`udid`) of the drawing package resource.

Cookies	Headers (9)	Test Results
KEY	VALUE	
Content-Length ⓘ	243	
Content-Type ⓘ	application/json; charset=utf-8	
x-ms-azuremaps-reg... ⓘ	West US 2	
X-Content-Type-Opti... ⓘ	nosniff	
Strict-Transport-Sec... ⓘ	max-age=31536000; includeSubDomains	
Resource-Location ⓘ	https://us.atlas.microsoft.com/mapData/metadata/6ebf1ae1-2a66-760b-e28c-b9381fcff335?api-version=2.0	
X-Cache ⓘ	CONFIG_NOCACHE	
X-MSEdge-Ref ⓘ	Ref A: 145D7FD71F604535B6952C25E019A227 Ref B: WSTEDGE1120 Ref C: 2021-05-20T16:38:20Z	
Date ⓘ	Thu, 20 May 2021 16:38:20 GMT	

(Optional) Retrieve Drawing package metadata

You can retrieve metadata from the Drawing package resource. The metadata contains information like the resource location URL, creation date, updated date, size, and upload status.

To retrieve content metadata:

1. Select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name** for the request, such as *GET Data Upload Status*.
4. Select the collection you previously created, and then select **Save**.
5. Select the **GET** HTTP method.
6. Enter the **resource Location URL** you copied in [Check Drawing package upload status](#). The request should look like the following URL (replace **{Azure-Maps-Primary-Subscription-key}** with your primary subscription key):

```
https://us.atlas.microsoft.com/mapData/metadata/{udid}?api-version=2.0&subscription-key={Azure-Maps-Primary-Subscription-key}
```

7. Select **Send**.
8. In the response window, select the **Body** tab. The metadata should like the following JSON fragment:

```
{
    "udid": "{udid}",
    "location": "https://us.atlas.microsoft.com/mapData/6ebf1ae1-2a66-760b-e28c-b9381fcff335?api-version=2.0",
    "created": "5/18/2021 8:10:32 PM +00:00",
    "updated": "5/18/2021 8:10:37 PM +00:00",
    "sizeInBytes": 946901,
    "uploadStatus": "Completed"
}
```

Convert a Drawing package

Now that the Drawing package is uploaded, we'll use the **udid** for the uploaded package to convert the package into map data. The Conversion API uses a long-running transaction that implements the pattern defined [here](#).

To convert a Drawing package:

1. Select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name** for the request, such as *POST Convert Drawing Package*.
4. Select the collection you previously created, and then select **Save**.
5. Select the **POST** HTTP method.
6. Enter the following URL to the [Conversion Service](#) (replace **{Azure-Maps-Primary-Subscription-key}** with your primary subscription key and **udid** with the **udid** of the uploaded package):

```
https://us.atlas.microsoft.com/conversions?subscription-key={Azure-Maps-Primary-Subscription-key}&api-version=2.0&udid={udid}&inputType=DWG&outputOntology=facility-2.0
```

7. Select **Send**.
8. In the response window, select the **Headers** tab.

9. Copy the value of the **Operation-Location** key, which is the `status URL`. We'll use the `status URL` to check the status of the conversion.

Body	Cookies	Headers (9)	Test Results
			Status: 202 Accepted Time: 14
KEY	VALUE		
Content-Type ⓘ	application/json		
x-ms-azuremaps-region ⓘ	West US 2		
X-Content-Type-Options ⓘ	nosniff		
Strict-Transport-Security ⓘ	max-age=31536000; includeSubDomains		
Operation-Location ⓘ	https://us.atlas.microsoft.com/conversions/operations/c1414013-4677-4d6f-bce4-eb65f14c237f?api-version=2.0		
X-Cache ⓘ	CONFIG_NOCACHE		
X-MSEdge-Ref ⓘ	Ref A: B5090DE2CEB84A2CA9EEE2EC07537853 Ref B: WSTEDGE1116 Ref C: 2021-05-19T18:24:27Z		
Date ⓘ	Wed, 19 May 2021 18:24:28 GMT		
Content-Length ⓘ	0		

Check the Drawing package conversion status

After the conversion operation completes, it returns a `conversionId`. We can access the `conversionId` by checking the status of the Drawing package conversion process. The `conversionId` can then be used to access the converted data.

To check the status of the conversion process and retrieve the `conversionId`:

1. Select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name** for the request, such as *GET Conversion Status*.
4. Select the collection you previously created, and then select **Save**.
5. Select the **GET** HTTP method:
6. Enter the `status URL` you copied in [Convert a Drawing package](#). The request should look like the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key):

```
https://us.atlas.microsoft.com/conversions/operations/<operationId>?api-version=2.0&subscription-key={Azure-Maps-Primary-Subscription-key}
```

7. Select **Send**.
8. In the response window, select the **Headers** tab.
9. Copy the value of the **Resource-Location** key, which is the `resource location URL`. The `resource location URL` contains the unique identifier (`conversionId`), which can be used by other APIs to access the converted map data.

Headers (9)		Test Results
KEY	VALUE	
Content-Length	593	
Content-Type	application/json; charset=utf-8	
x-ms-azurermaps-regi...	West US 2	
X-Content-Type-Opti...	nosniff	
Strict-Transport-Secu...	max-age=31536000; includeSubDomains	
Resource-Location	https://us.atlas.microsoft.com/conversions/ 5dd13c86-5dbd-b058-1024-e6f392f40baa ?api-version=2.0	
X-Cache	CONFIG_NOCACHE	
X-MSEdge-Ref	Ref A: 26478BDDA9534CB9ABE456DB40020736 Ref B: WSTEDGE1012 Ref C: 2021-05-20T17:15:50Z	
Date	Thu, 20 May 2021 17:15:50 GMT	

The sample Drawing package should be converted without errors or warnings. However, if you receive errors or warnings from your own Drawing package, the JSON response includes a link to the [Drawing error visualizer](#). You can use the Drawing Error visualizer to inspect the details of errors and warnings. To receive recommendations to resolve conversion errors and warnings, see [Drawing conversion errors and warnings](#).

The following JSON fragment displays a sample conversion warning:

```
{
    "operationId": "<operationId>",
    "created": "2021-05-19T18:24:28.7922905+00:00",
    "status": "Succeeded",
    "warning": {
        "code": "dwgConversionProblem",
        "details": [
            {
                "code": "warning",
                "details": [
                    {
                        "code": "manifestWarning",
                        "message": "Ignoring unexpected JSON property:
unitProperties[0].nonWheelchairAccessible with value False"
                    }
                ]
            }
        ],
        "properties": {
            "diagnosticPackageLocation": "https://atlas.microsoft.com/mapData/ce61c3c1-faa8-75b7-349f-d863f6523748?api-version=1.0"
        }
    }
}
```

Create a dataset

A dataset is a collection of map features, such as buildings, levels, and rooms. To create a dataset, use the [Dataset Create API](#). The Dataset Create API takes the `conversionId` for the converted Drawing package and returns a `datasetId` of the created dataset.

To create a dataset:

1. Select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name** for the request, such as *POST Dataset Create*.
4. Select the collection you previously created, and then select **Save**.

- Select the **POST** HTTP method.
- Enter the following URL to the [Dataset API](#). The request should look like the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key, and `{conversionId}` with the `conversionId` obtained in [Check Drawing package conversion status](#)):

```
https://us.atlas.microsoft.com/datasets?api-version=2.0&conversionId={conversionId}&type=facility&subscription-key={Azure-Maps-Primary-Subscription-key}
```

- Select **Send**.
- In the response window, select the **Headers** tab.
- Copy the value of the **Operation-Location** key, which is the `status URL`. We'll use the `status URL` to check the status of the dataset.

Cookies Headers (9) Test Results

Status: 202

KEY	VALUE
Content-Type ⓘ	application/json
x-ms-azurermaps-region ⓘ	West US 2
X-Content-Type-Options ⓘ	nosniff
Strict-Transport-Security ⓘ	max-age=31536000; includeSubDomains
Operation-Location ⓘ	https://us.atlas.microsoft.com/datasets/operations/92267b58-1078-4212-8fe2-0266fb3e9bd8?api-version=2.0
X-Cache ⓘ	CONFIG_NOCACHE
X-MSEdge-Ref ⓘ	Ref A: 9E9CBE907A544F8D890B44497656B2F8 Ref B: WSTEDGE0920 Ref C: 2021-05-19T19:19:21Z
Date ⓘ	Wed, 19 May 2021 19:19:21 GMT
Content-Length ⓘ	0

Check the dataset creation status

To check the status of the dataset creation process and retrieve the `datasetId`:

- Select **New**.
- In the **Create New** window, select **Request**.
- Enter a **Request name** for the request, such as *GET Dataset Status*.
- Select the collection you previously created, and then select **Save**.
- Select the **GET** HTTP method.
- Enter the `status URL` you copied in [Create a dataset](#). The request should look like the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key):

```
https://us.atlas.microsoft.com/datasets/operations/<operationId>?api-version=2.0&subscription-key={Azure-Maps-Primary-Subscription-key}
```

- Select **Send**.
- In the response window, select the **Headers** tab. The value of the **Resource-Location** key is the `resource location URL`. The `resource location URL` contains the unique identifier (`datasetId`) of the dataset.
- Copy the `datasetId`, because you'll use it in the next sections of this tutorial.

Body	Cookies	Headers (9)	Test Results	Save Response
KEY	VALUE			
Content-Length	235			
Content-Type	application/json; charset=utf-8			
x-ms-azuremaps-region	West US 2			
X-Content-Type-Options	nosniff			
Strict-Transport-Security	max-age=31536000; includeSubDomains			
Resource-Location	https://us.atlas.microsoft.com/datasets/02eaf621-dfcc-e90f-c887-f4e459295942?api-version=2.0			
X-Cache	CONFIG_NOCACHE			
X-MSEdge-Ref	Ref A: 95502151BF2B4B84A7E70834219DECA4 Ref B: WSTEDGE1012 Ref C: 2021-05-20T17:26:24Z			
Date	Thu, 20 May 2021 17:26:24 GMT			

Create a tileset

A tileset is a set of vector tiles that render on the map. Tilesets are created from existing datasets. However, a tileset is independent from the dataset from which it was sourced. If the dataset is deleted, the tileset continues to exist.

To create a tileset:

1. Select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name** for the request, such as *POST Tileset Create*.
4. Select the collection you previously created, and then select **Save**.
5. Select the **POST** HTTP method.
6. Enter the following URL to the [Tileset API](#). The request should look like the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key), and `{datasetId}` with the `datasetId` obtained in [Check dataset creation status](#):

```
https://us.atlas.microsoft.com/tilesets?api-version=2.0&datasetID={datasetId}&subscription-key={Azure-Maps-Primary-Subscription-key}
```

7. Select **Send**.
8. In the response window, select the **Headers** tab.
9. Copy the value of the **Operation-Location** key, which is the `status URL`. We'll use the `status URL` to check the status of the tileset.

Cookies	Headers (9)	Test Results
KEY	VALUE	
Content-Type ⓘ	application/json	
x-ms-azurermaps-region ⓘ	West US 2	
X-Content-Type-Options ⓘ	nosniff	
Strict-Transport-Security ⓘ	max-age=31536000; includeSubDomains	
Operation-Location ⓘ	https://us.atlas.microsoft.com/tilesets/operations/cab3a701-b404-4501-966a-1d4862a2619d?api-version=2.0	
X-Cache ⓘ	CONFIG_NOCACHE	
X-MSEdge-Ref ⓘ	Ref A: 76763AB2649C4B8B88650A7357BD3C03 Ref B: WSTEDGE0917 Ref C: 2021-05-19T19:29:32Z	
Date ⓘ	Wed, 19 May 2021 19:29:35 GMT	
Content-Length ⓘ	0	

Check the tileset creation status

To check the status of the dataset creation process and retrieve the `tilesetId`:

1. Select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name** for the request, such as *GET Tileset Status*.
4. Select the collection you previously created, and then select **Save**.
5. Select the **GET** HTTP method.
6. Enter the `status URL` you copied in [Create a tileset](#). The request should look like the following URL
(replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key):

```
https://us.atlas.microsoft.com/tilesets/operations/<operationId>?api-version=2.0&subscription-key={Azure-Maps-Primary-Subscription-key}
```

7. Select **Send**.
8. In the response window, select the **Headers** tab. The value of the **Resource-Location** key is the `resource location URL`. The `resource location URL` contains the unique identifier (`tilesetId`) of the dataset.

Cookies	Headers (9)	Test Results
KEY	VALUE	
Content-Length ⓘ	235	
Content-Type ⓘ	application/json; charset=utf-8	
x-ms-azurermaps-region ⓘ	West US 2	
X-Content-Type-Options ⓘ	nosniff	
Strict-Transport-Security ⓘ	max-age=31536000; includeSubDomains	
Resource-Location ⓘ	https://us.atlas.microsoft.com/tilesets/d8eed547-dcc7-bf52-5b75-082bcd7c5ac1?api-version=2.0	
X-Cache ⓘ	CONFIG_NOCACHE	
X-MSEdge-Ref ⓘ	Ref A: D4117D7DDCEA4490975B653F92CE235F Ref B: WSTEDGE0621 Ref C: 2021-05-20T17:29:38Z	
Date ⓘ	Thu, 20 May 2021 17:29:38 GMT	

Query datasets with WFS API

Datasets can be queried using [WFS API](#). You can use the WFS API to query for all feature collections or a specific collection. In this section of the tutorial, we'll do both. First we'll query all collections, and then we will query for the `unit` collection.

Query for feature collections

To query the all collections in your dataset:

1. Select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name** for the request, such as *GET Dataset Collections*.
4. Select the collection you previously created, and then select **Save**.
5. Select the **GET** HTTP method.
6. Enter the following URL to [WFS API](#). The request should look like the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key), and `{datasetId}` with the `datasetId` obtained in [Check dataset creation status](#):

```
https://us.atlas.microsoft.com/wfs/datasets/{datasetId}/collections?subscription-key={Azure-Maps-Primary-Subscription-key}&api-version=2.0
```

7. Select **Send**.
8. The response body is returned in GeoJSON format and contains all collections in the dataset. For simplicity, the example here only shows the `unit` collection. To see an example that contains all collections, see [WFS Describe Collections API](#). To learn more about any collection, you can select any of the URLs inside the `link` element.

```
{
  "collections": [
    {
      "name": "unit",
      "description": "A physical and non-overlapping area which might be occupied and traversed by a navigating agent. Can be a hallway, a room, a courtyard, etc. It is surrounded by physical obstruction (wall), unless the is_open_area attribute is equal to true, and one must add openings where the obstruction shouldn't be there. If is_open_area attribute is equal to true, all the sides are assumed open to the surroundings and walls are to be added where needed. Walls for open areas are represented as a line_element or area_element with is_obstruction equal to true.",
      "links": [
        {
          "href": "https://atlas.microsoft.com/wfs/datasets/{datasetId}/collections/unit/definition?api-version=1.0",
          "rel": "describedBy",
          "title": "Metadata catalogue for unit"
        },
        {
          "href": "https://atlas.microsoft.com/wfs/datasets/{datasetId}/collections/unit/items?api-version=1.0",
          "rel": "data",
          "title": "unit"
        },
        {
          "href": "https://atlas.microsoft.com/wfs/datasets/{datasetId}/collections/unit?api-version=1.0",
          "rel": "self",
          "title": "Metadata catalogue for unit"
        }
      ]
    },
  ]
},
```

Query for unit feature collection

In this section, we'll query [WFS API](#) for the `unit` feature collection.

To query the unit collection in your dataset:

1. Select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name** for the request, such as *GET Unit Collection*.
4. Select the collection you previously created, and then select **Save**.
5. Select the **GET** HTTP method.
6. Enter the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key, and `{datasetId}` with the `datasetId` obtained in [Check dataset creation status](#)):

```
https://us.atlas.microsoft.com/wfs/datasets/{datasetId}/collections/unit/items?subscription-key={Azure-Maps-Primary-Subscription-key}&api-version=2.0
```

7. Select **Send**.

8. After the response returns, copy the feature `id` for one of the `unit` features. In the following example, the feature `id` is "UNIT26". In this tutorial, we'll use "UNIT26" as our feature `id` in the next section.

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [...]
      },
      "properties": {
        "original_id": "b7410920-8cb0-490b-ab23-b489fd35aed0",
        "category_id": "CTG8",
        "is_open_area": true,
        "navigable_by": [
          "pedestrian"
        ],
        "route_through_behavior": "allowed",
        "level_id": "LVL14",
        "occupants": [],
        "address_id": "DIR1",
        "name": "157"
      },
      "id": "UNIT26",
      "featureType": ""
    }, {"..."}
  ]
}
```

Create a feature stateset

Feature statesets define dynamic properties and values on specific features that support them. In this section, we'll create a stateset that defines boolean values and corresponding styles for the **occupancy** property.

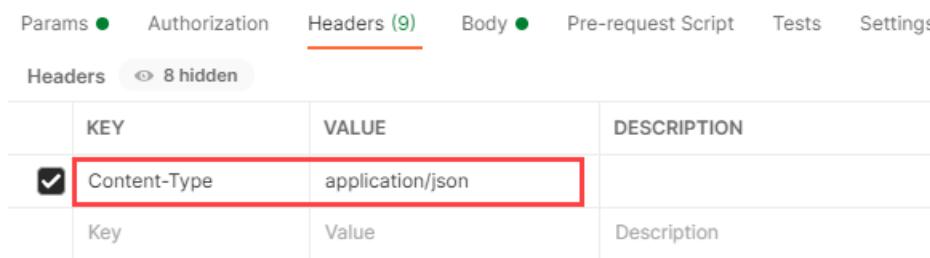
To create a stateset:

1. Select **New**.
2. In the **Create New** window, select **Request**.

3. Enter a **Request name** for the request, such as *POST Create Stateset*.
4. Select the collection you previously created, and then select **Save**.
5. Select the **POST** HTTP method.
6. Enter the following URL to the [Stateset API](#). The request should look like the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key, and `{datasetId}` with the `datasetId` obtained in [Check dataset creation status](#)):

```
https://us.atlas.microsoft.com/featurestatesets?api-version=2.0&datasetId={datasetId}&subscription-key={Azure-Maps-Primary-Subscription-key}
```

7. Select the **Headers** tab.
8. In the **KEY** field, select `Content-Type`.
9. In the **VALUE** field, select `application/json`.



The screenshot shows the Postman interface with the 'Headers' tab selected. There is one entry in the table:

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	Content-Type	application/json	
	Key	Value	Description

10. Select the **Body** tab.
11. In the dropdown lists, select **raw** and **JSON**.
12. Copy the following JSON styles, and then paste them in the **Body** window:

```
{
  "styles": [
    {
      "keyname": "occupied",
      "type": "boolean",
      "rules": [
        {
          "true": "#FF0000",
          "false": "#00FF00"
        }
      ]
    }
  ]
}
```

13. Select **Send**.
14. After the response returns successfully, copy the `statesetId` from the response body. In the next section, we'll use the `statesetId` to change the `occupancy` property state of the unit with feature `id` "UNIT26".

Pretty

Raw

Preview

Visualize

JSON



```

1  {
2   "statesetId": "4b8c219e-763a-ff7d-be61-d94e3115be45"
3 }
```

Update a feature state

To update the `occupied` state of the unit with feature `id` "UNIT26":

1. Select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name** for the request, such as *POST Set Stateset*.
4. Select the collection you previously created, and then select **Save**.
5. Select the **PUT** HTTP method.
6. Enter the following URL to the [Feature Statesets API](#). The request should look like the following URL
 (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key, and `{statesetId}` with the `statesetId` obtained in [Create a feature stateset](#)):

```
https://us.atlas.microsoft.com/featurestatesets/{statesetId}/featureStates/UNIT26?api-version=2.0&subscription-key={Azure-Maps-Primary-Subscription-key}
```

7. Select the **Headers** tab.
8. In the **KEY** field, select `Content-Type`.
9. In the **VALUE** field, select `application/json`.

Params ● Authorization Headers (9) Body ● Pre-request Script Tests Settings

Headers (8 hidden)

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	Content-Type	application/json	
	Key	Value	Description

10. Select the **Body** tab.
11. In the dropdown lists, select **raw** and **JSON**.
12. Copy the following JSON style, and then paste it in the **Body** window:

```
{  
  "states": [  
    {  
      "keyName": "occupied",  
      "value": true,  
      "eventTimestamp": "2020-11-14T17:10:20"  
    }  
  ]  
}
```

NOTE

The update will be saved only if the time posted stamp is after the time stamp of the previous request.

13. Select **Send**.

14. After the update completes, you'll receive a `200 OK` HTTP status code. If you implemented [dynamic styling](#) for an indoor map, the update displays at the specified time stamp in your rendered map.

You can use the [Feature Get Stateset API](#) to retrieve the state of a feature using its feature `id`. You can also use the [Feature State Delete State API](#) to delete the stateset and its resources.

To learn more about the different Azure Maps Creator services discussed in this article, see [Creator Indoor Maps](#).

Clean up resources

There aren't any resources that require cleanup.

Next steps

To learn how to use the Indoor Maps module, see

[Use the Indoor Maps module](#)

Tutorial: Set up a geofence by using Azure Maps

6/1/2021 • 12 minutes to read • [Edit Online](#)

This tutorial walks you through the basics of creating and using Azure Maps geofence services. You'll do this in the context of the following scenario:

A construction site manager must track equipment as it enters and leaves the perimeters of a construction area. Whenever a piece of equipment exits or enters these perimeters, an email notification is sent to the operations manager.

Azure Maps provides a number of services to support the tracking of equipment entering and exiting the construction area. In this tutorial, you:

- Upload [Geofencing GeoJSON data](#) that defines the construction site areas you want to monitor. You'll use the [Data Upload API](#) to upload geofences as polygon coordinates to your Azure Maps account.
- Set up two [logic apps](#) that, when triggered, send email notifications to the construction site operations manager when equipment enters and exits the geofence area.
- Use [Azure Event Grid](#) to subscribe to enter and exit events for your Azure Maps geofence. You set up two webhook event subscriptions that call the HTTP endpoints defined in your two logic apps. The logic apps then send the appropriate email notifications of equipment moving beyond or entering the geofence.
- Use [Search Geofence Get API](#) to receive notifications when a piece of equipment exits and enters the geofence areas.

Prerequisites

1. [Create an Azure Maps account](#).
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.

This tutorial uses the [Postman](#) application, but you can choose a different API development environment.

Upload geofencing GeoJSON data

For this tutorial, you upload geofencing GeoJSON data that contains a `FeatureCollection`. The `FeatureCollection` contains two geofences that define polygonal areas within the construction site. The first geofence has no time expiration or restrictions. The second one can only be queried against during business hours (9:00 AM-5:00 PM in the Pacific Time zone), and will no longer be valid after January 1, 2022. For more information on the GeoJSON format, see [Geofencing GeoJSON data](#).

TIP

You can update your geofencing data at any time. For more information, see [Data Upload API](#).

1. Open the Postman app. Near the top, select **New**. In the **Create New** window, select **Collection**. Name the collection and select **Create**.
2. To create the request, select **New** again. In the **Create New** window, select **Request**. Enter a **Request name** for the request. Select the collection you created in the previous step, and then select **Save**.
3. Select the **POST** HTTP method in the builder tab, and enter the following URL to upload the geofencing data to Azure Maps. For this request, and other requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key.

```
https://us.atlas.microsoft.com/mapData?subscription-key={Azure-Maps-Primary-Subscription-key}&api-version=2.0&dataFormat=geojson
```

The `geojson` parameter in the URL path represents the data format of the data being uploaded.

4. Select the **Body** tab. Select **raw**, and then **JSON** as the input format. Copy and paste the following GeoJSON data into the **Body** text area:

```
{  
  "type": "FeatureCollection",  
  "features": [  
    {  
      "type": "Feature",  
      "geometry": {  
        "type": "Polygon",  
        "coordinates": [  
          [  
            [  
              [-122.13393688201903,  
               47.63829579223815  
              ],  
              [  
                [-122.13389128446579,  
                 47.63782047131512  
                ],  
                [  
                  [-122.13240802288054,  
                   47.63783312249837  
                  ],  
                  [  
                    [-122.13238388299942,  
                     47.63829037035086  
                    ],  
                    [  
                      [-122.13393688201903,  
                       47.63829579223815  
                      ]  
                    ]  
                  ]  
                ]  
              ]  
            ]  
          ]  
        ]  
      },  
      "properties": {  
        "geometryId": "1"  
      }  
    },  
    {  
      "type": "Feature",  
      "geometry": {  
        "type": "Polygon",  
        "coordinates": [  
          [  
            [  
              [-122.13374376296996,  
               47.63784758098976  
              ],  
              [  
                [-122.13277012109755,  
                 47.63784577367854  
                ],  
                [  
                  [-122.13314831256866,  
                   47.6382813338708  
                  ],  
                  [  
                    [-122.1334782242775,  
                     47.63827591198201  
                    ]  
                  ]  
                ]  
              ]  
            ]  
          ]  
        ]  
      }  
    }  
  ]  
}
```

```

        ],
        [
          -122.13374376296996,
          47.63784758098976
        ]
      ],
      "properties": {
        "geometryId": "2",
        "validityTime": {
          "expiredTime": "2022-01-01T00:00:00",
          "validityPeriod": [
            {
              "start Time": "2020-07-15T16:00:00",
              "end Time": "2020-07-15T24:00:00",
              "recurrenceType": "Daily",
              "recurrenceFrequency": 1,
              "businessDayOnly": true
            }
          ]
        }
      }
    ]
  }
}

```

- Select **Send**, and wait for the request to process. When the request completes, go to the **Headers** tab of the response. Copy the value of the **Operation-Location** key, which is the `status URL`.

```
https://us.atlas.microsoft.com/mapData/operations/<operationId>?api-version=2.0
```

- To check the status of the API call, create a **GET** HTTP request on the `status URL`. You'll need to append your primary subscription key to the URL for authentication. The **GET** request should like the following URL:

```
https://us.atlas.microsoft.com/mapData/<operationId>?api-version=2.0&subscription-key={Subscription-key}
```

- When the request completes successfully, select the **Headers** tab in the response window. Copy the value of the **Resource-Location** key, which is the `resource location URL`. The `resource location URL` contains the unique identifier (`udid`) of the uploaded data. Save the `udid` to query the Get Geofence API in the last section of this tutorial. Optionally, you can use the `resource location URL` to retrieve metadata from this resource in the next step.

Headers (9)		Test Results	Status: 200 OK	Time: 395 ms	Size: 746 B	Save Response
KEY	VALUE					
Content-Length	243					
Content-Type	application/json; charset=utf-8					
x-ms-azuremaps-reg...	West US 2					
X-Content-Type-Opti...	nosniff					
Strict-Transport-Sec...	max-age=31536000; includeSubDomains					
Resource-Location	<code>https://us.atlas.microsoft.com/mapData/metadata/6ebf1ae1-2a66-760b-e28c-b9381fcff335?api-version=2.0</code>					
X-Cache	CONFIG_NOCACHE					
X-MSEdge-Ref	Ref A: 145D7FD71F604535B6952C25E019A227 Ref B: WSTEDGE1120 Ref C: 2021-05-20T16:38:20Z					
Date	Thu, 20 May 2021 16:38:20 GMT					

- To retrieve content metadata, create a **GET** HTTP request on the `resource location URL` that was retrieved

in step 7. Make sure to append your primary subscription key to the URL for authentication. The GET request should like the following URL:

```
https://us.atlas.microsoft.com/mapData/metadata/{udid}?api-version=2.0&subscription-key={Azure-Maps-Primary-Subscription-key}
```

- When the request completes successfully, select the **Headers** tab in the response window. The metadata should like the following JSON fragment:

```
{  
    "udid": "{udid}",  
    "location": "https://us.atlas.microsoft.com/mapData/6ebf1ae1-2a66-760b-e28c-b9381fcff335?api-version=2.0",  
    "created": "5/18/2021 8:10:32 PM +00:00",  
    "updated": "5/18/2021 8:10:37 PM +00:00",  
    "sizeInBytes": 946901,  
    "uploadStatus": "Completed"  
}
```

Create workflows in Azure Logic Apps

Next, you create two [logic app](#) endpoints that trigger an email notification. Here's how to create the first one:

- Sign in to the [Azure portal](#).
- In the upper-left corner of the Azure portal, select **Create a resource**.
- In the **Search the Marketplace** box, type **Logic App**.
- From the results, select **Logic App > Create**.
- On the **Logic App** page, enter the following values:
 - The **Subscription** that you want to use for this logic app.
 - The **Resource group** name for this logic app. You can choose to **Create new** or **Use existing** resource group.
 - The **Logic App name** of your logic app. In this case, use `Equipment-Enter` as the name.

For the purposes of this tutorial, keep all other values on their default settings.

Logic App - Microsoft Azure

https://ms.portal.azure.com/#create/Microsoft.Empty...

Microsoft Azure (Preview)

Search resources, services, and docs (G+/)

Home > New > Logic App >

Logic App

* Basics Tags Review + create

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * contoso-subscription

Resource group * (New) contoso-resource [Create new](#)

Instance details

Logic App name * Equipment-Enter

Select the location Region Integration Service Environment

Location * East US

Log Analytics [On](#) [Off](#)

[Review + create](#) [Previous : Basics](#) [Next : Tags >](#) [Download a template for automation](#)

The screenshot shows the 'Logic App' creation page in the Microsoft Azure portal. It's on the 'Basics' tab. Under 'Project details', the subscription is set to 'contoso-subscription' and the resource group is '(New) contoso-resource'. In the 'Instance details' section, the logic app name is 'Equipment-Enter', it's set to 'Region' for location, and the location is 'East US'. Log Analytics is turned off. At the bottom, there are navigation links for 'Review + create', 'Previous : Basics', 'Next : Tags >', and 'Download a template for automation'.

6. Select **Review + Create**. Review your settings and select **Create** to submit the deployment. When the deployment successfully completes, select **Go to resource**. You're taken to **Logic App Designer**.
7. Select a trigger type. Scroll down to the **Start with a common trigger** section. Select **When an HTTP request is received**.

Logic Apps Designer - Microsoft

https://ms.portal.azure.com/#@microsoft.onmicrosoft.com

Microsoft Azure (Preview)

Search resources, services, and docs (G+)

Home > Microsoft.EmptyWorkflow | Overview > contoso-logicapp >

Logic Apps Designer

Start with a common trigger
Pick from one of the most commonly used triggers, then orchestrate any number of actions using the rich collection of connectors

 When a message is received in a Service Bus queue	 When a HTTP request is received
 When a new tweet is posted	 When an Event Grid resource event occurs
 Recurrence	 When a new email is received in Outlook.com
 When a new file is created on OneDrive	 When a file is added to FTP server

8. In the upper-right corner of Logic App Designer, select **Save**. The **HTTP POST URL** is automatically generated. Save the URL. You need it in the next section to create an event endpoint.

When a HTTP request is received

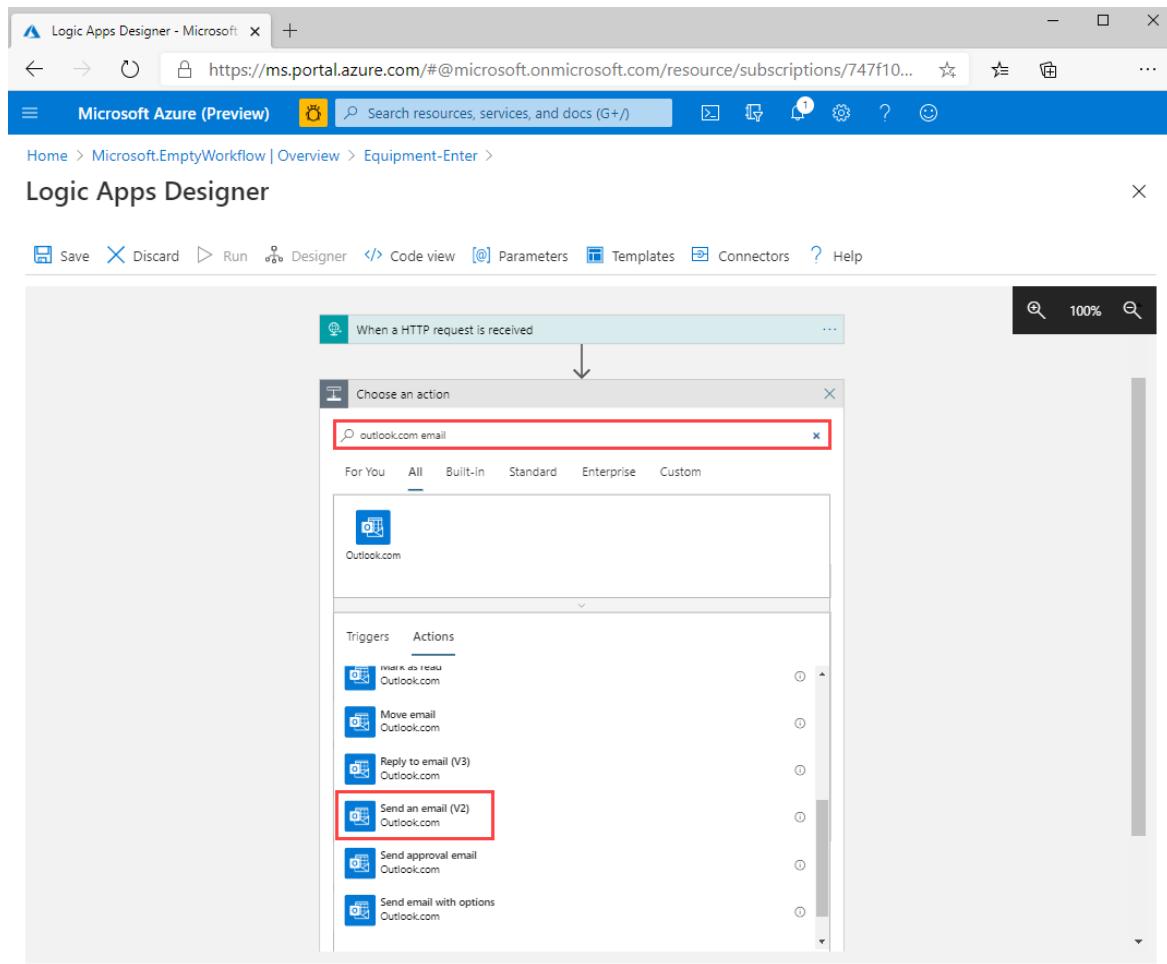
HTTP POST URL
<https://prod-47.eastus.logic.azure.com:443/workflows/671631872f534848aee911d7917217bd/trig...>

Request Body JSON Schema

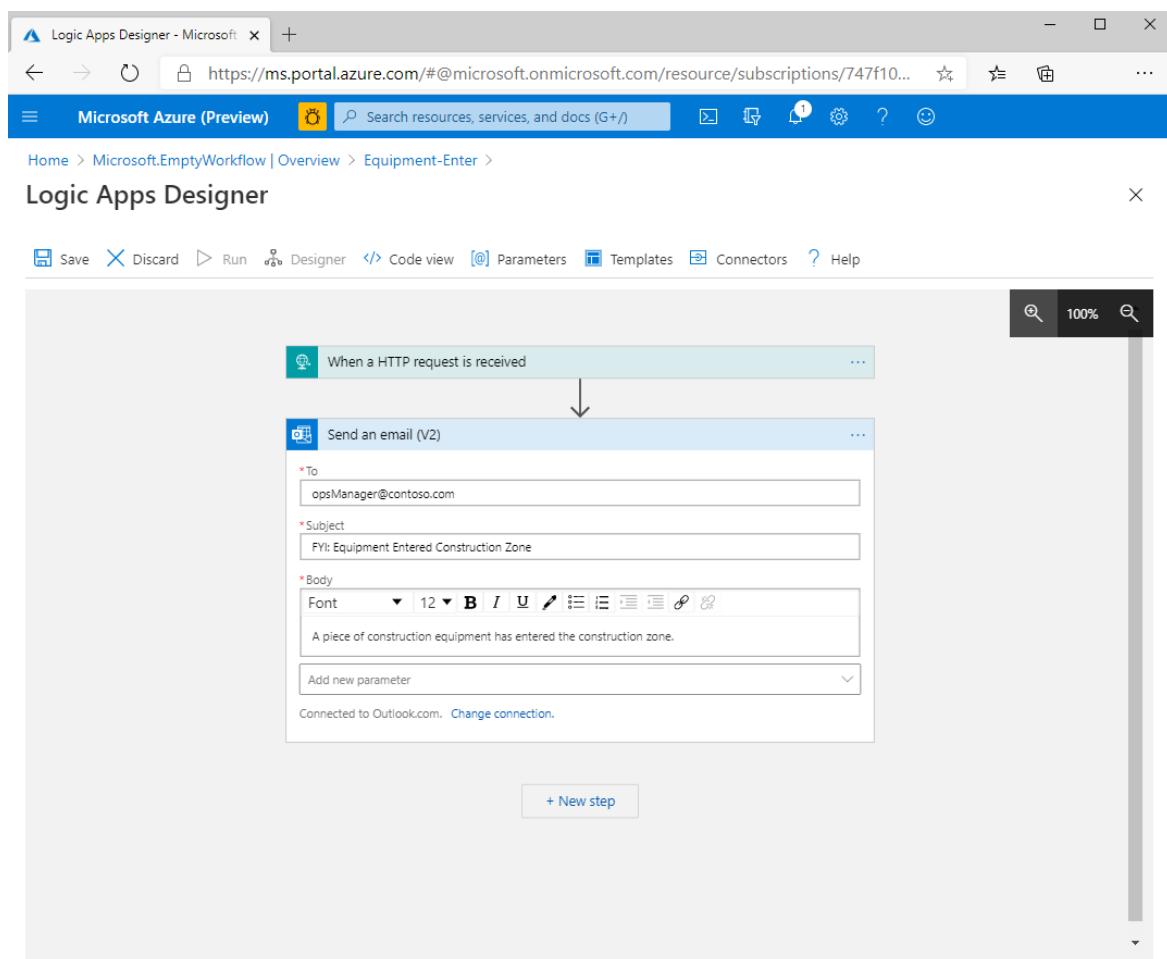
Use sample payload to generate schema

Add new parameter

9. Select **+ New Step**. Now you'll choose an action. Type `outlook.com email` in the search box. In the **Actions** list, scroll down and select **Send an email (V2)**.



10. Sign in to your Outlook account. Make sure to select **Yes** to allow the logic app to access the account. Fill in the fields for sending an email.



TIP

You can retrieve GeoJSON response data, such as `geometryId` or `deviceId`, in your email notifications. You can configure Logic Apps to read the data sent by Event Grid. For information on how to configure Logic Apps to consume and pass event data into email notifications, see [Tutorial: Send email notifications about Azure IoT Hub events using Event Grid and Logic Apps](#).

11. In the upper-left corner of Logic App Designer, select **Save**.

To create a second logic app to notify the manager when equipment exits the construction site, repeat steps 3-

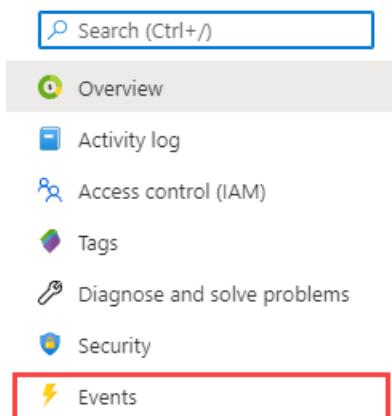
11. Name the logic app `Equipment-Exit`.

Create Azure Maps events subscriptions

Azure Maps supports [three event types](#). Here, you need to create two different event subscriptions: one for geofence enter events, and one for geofence exit events.

The following steps show how to create an event subscription for the geofence enter events. You can subscribe to geofence exit events by repeating the steps in a similar manner.

1. Go to your Azure Maps account. In the dashboard, select **Subscriptions**. Select your subscription name, and select **events** from the settings menu.



2. To create an event subscription, select **+ Event Subscription** from the events page.



3. On the **Create Event Subscription** page, enter the following values:

- The **Name** of the event subscription.
- The **Event Schema** should be *Event Grid Schema*.
- The **System Topic Name** for this event subscription, which in this case is `Contoso-Construction`.
- For **Filter to Event Types**, choose `Geofence Entered` as the event type.
- For **Endpoint Type**, choose `Web Hook`.
- For **Endpoint**, copy the HTTP POST URL for the logic app enter endpoint that you created in the previous section. If you forgot to save it, you can just go back into Logic App Designer and copy it from the HTTP trigger step.

The screenshot shows the 'Create Event Subscription' page in the Microsoft Azure (Preview) portal. The page is titled 'Create Event Subscription' under the 'Event Grid' section. It has tabs for 'Basic', 'Filters', 'Additional Features', and 'Advanced Editor'. The 'Basic' tab is selected.

EVENT SUBSCRIPTION DETAILS

Name: Equipment-Enter-Events

Event Schema: Event Grid Schema

TOPIC DETAILS

Topic Type: Azure Maps Account

Source Resource: contoso-maps

System Topic Name: Contoso-Construction

EVENT TYPES

Filter to Event Types: Geofence Entered

ENDPOINT DETAILS

Endpoint Type: Web Hook (change)

Endpoint: https://prod-50.eastus.logic.azure.com:443/workflows/77a5b... (change)

Create

4. Select **Create**.

Repeat steps 1-4 for the logic app exit endpoint that you created in the previous section. On step 3, make sure to choose **Geofence Exited** as the event type.

Use Spatial Geofence Get API

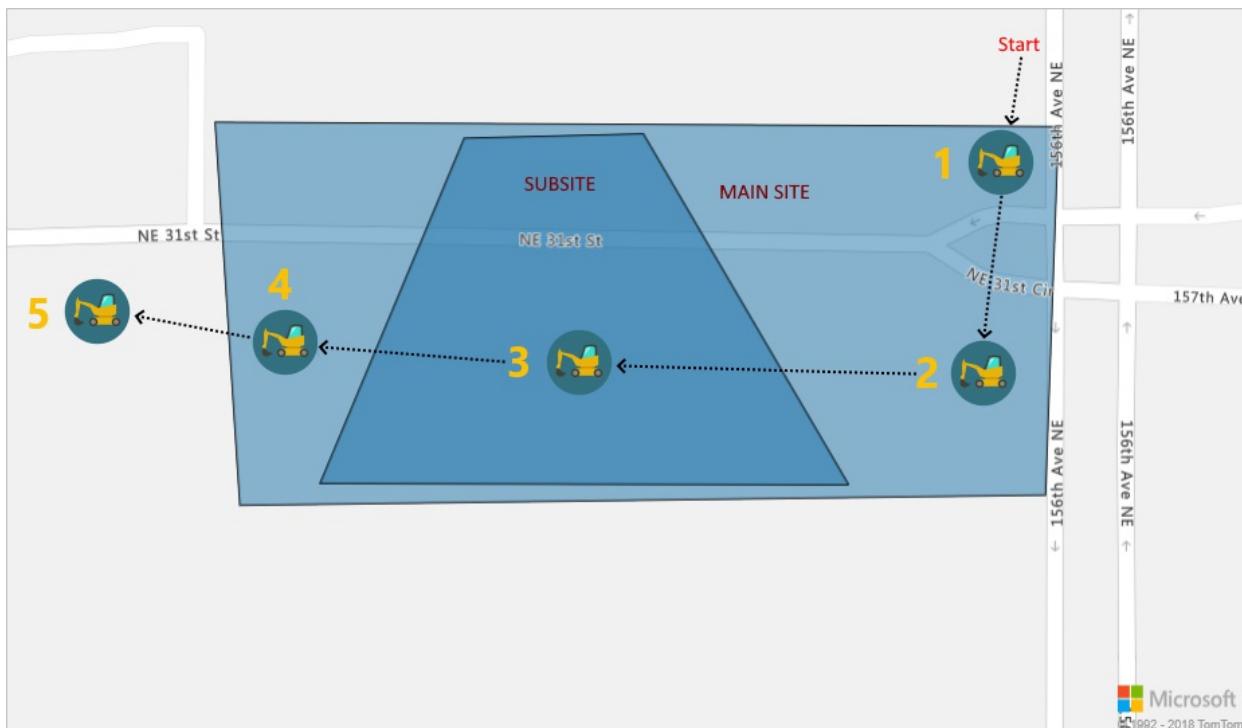
Use [Spatial Geofence Get API](#) to send email notifications to the operations manager when a piece of equipment enters or exits the geofences.

Each piece of equipment has a `deviceId`. In this tutorial, you're tracking a single piece of equipment, with a unique ID of `device_1`.

The following diagram shows the five locations of the equipment over time, beginning at the *Start* location, which is somewhere outside the geofences. For the purposes of this tutorial, the *Start* location is undefined, because you won't query the device at that location.

When you query the [Spatial Geofence Get API](#) with an equipment location that indicates initial geofence entry or exit, Event Grid calls the appropriate logic app endpoint to send an email notification to the operations manager.

Each of the following sections makes API requests by using the five different location coordinates of the equipment.



Equipment location 1 (47.638237,-122.132483)

1. Near the top of the Postman app, select **New**. In the **Create New** window, select **Request**. Enter a **Request name** for the request. Make it *Location 1*. Select the collection you created in the [Upload Geofencing GeoJSON data section](#), and then select **Save**.
2. Select the **GET** HTTP method in the builder tab, and enter the following URL. Make sure to replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key, and `{udid}` with the `udid` you saved in the [Upload Geofencing GeoJSON data section](#).

```
https://atlas.microsoft.com/spatial/geofence/json?subscription-key={subscription-key}&api-version=1.0&deviceId=device_01&udid={udid}&lat=47.638237&lon=-122.1324831&searchBuffer=5&isAsync=True&mode=EnterAndExit
```

3. Select **Send**. The following GeoJSON appears in the response window.

```
{
  "geometries": [
    {
      "deviceId": "device_1",
      "udId": "64f71aa5-bbee-942d-e351-651a6679a7da",
      "geometryId": "1",
      "distance": -999.0,
      "nearestLat": 47.638291,
      "nearestLon": -122.132483
    },
    {
      "deviceId": "device_1",
      "udId": "64f71aa5-bbee-942d-e351-651a6679a7da",
      "geometryId": "2",
      "distance": 999.0,
      "nearestLat": 47.638053,
      "nearestLon": -122.13295
    }
  ],
  "expiredGeofenceGeometryId": [],
  "invalidPeriodGeofenceGeometryId": [],
  "isEventPublished": true
}
```

In the preceding GeoJSON response, the negative distance from the main site geofence means that the equipment is inside the geofence. The positive distance from the subsite geofence means that the equipment is outside the subsite geofence. Because this is the first time this device has been located inside the main site geofence, the `isEventPublished` parameter is set to `true`. The operations manager receives an email notification that equipment has entered the geofence.

Location 2 (47.63800,-122.132531)

1. Near the top of the Postman app, select **New**. In the **Create New** window, select **Request**. Enter a **Request name** for the request. Make it *Location 2*. Select the collection you created in the [Upload Geofencing GeoJSON data section](#), and then select **Save**.
2. Select the **GET** HTTP method in the builder tab, and enter the following URL. Make sure to replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key, and `{udid}` with the `udid` you saved in the [Upload Geofencing GeoJSON data section](#).

```
https://atlas.microsoft.com/spatial/geofence/json?subscription-key={subscription-key}&api-version=1.0&deviceId=device_01&udId={udId}&lat=47.63800&lon=-122.132531&searchBuffer=5&isAsync=True&mode=EnterAndExit
```

3. Select **Send**. The following GeoJSON appears in the response window:

```
{
  "geometries": [
    {
      "deviceId": "device_01",
      "udId": "64f71aa5-bbee-942d-e351-651a6679a7da",
      "geometryId": "1",
      "distance": -999.0,
      "nearestLat": 47.637997,
      "nearestLon": -122.132399
    },
    {
      "deviceId": "device_01",
      "udId": "64f71aa5-bbee-942d-e351-651a6679a7da",
      "geometryId": "2",
      "distance": 999.0,
      "nearestLat": 47.63789,
      "nearestLon": -122.132809
    }
  ],
  "expiredGeofenceGeometryId": [],
  "invalidPeriodGeofenceGeometryId": [],
  "isEventPublished": false
}
```

In the preceding GeoJSON response, the equipment has remained in the main site geofence and hasn't entered the subsite geofence. As a result, the `isEventPublished` parameter is set to `false`, and the operations manager doesn't receive any email notifications.

Location 3 (47.63810783315048,-122.13336020708084)

1. Near the top of the Postman app, select **New**. In the **Create New** window, select **Request**. Enter a **Request name** for the request. Make it *Location 3*. Select the collection you created in the [Upload Geofencing GeoJSON data section](#), and then select **Save**.
2. Select the **GET** HTTP method in the builder tab, and enter the following URL. Make sure to replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key, and `{udid}` with the `udid` you saved in the [Upload Geofencing GeoJSON data section](#).

```
https://atlas.microsoft.com/spatial/geofence/json?subscription-key={subscription-key}&api-version=1.0&deviceId=device_01&udid={udid}&lat=47.63810783315048&lon=-122.13336020708084&searchBuffer=5&isAsync=True&mode=EnterAndExit
```

3. Select **Send**. The following GeoJSON appears in the response window:

```
{
  "geometries": [
    {
      "deviceId": "device_01",
      "udId": "64f71aa5-bbee-942d-e351-651a6679a7da",
      "geometryId": "1",
      "distance": -999.0,
      "nearestLat": 47.638294,
      "nearestLon": -122.133359
    },
    {
      "deviceId": "device_01",
      "udId": "64f71aa5-bbee-942d-e351-651a6679a7da",
      "geometryId": "2",
      "distance": -999.0,
      "nearestLat": 47.638161,
      "nearestLon": -122.133549
    }
  ],
  "expiredGeofenceGeometryId": [],
  "invalidPeriodGeofenceGeometryId": [],
  "isEventPublished": true
}
```

In the preceding GeoJSON response, the equipment has remained in the main site geofence, but has entered the subsite geofence. As a result, the `isEventPublished` parameter is set to `true`. The operations manager receives an email notification indicating that the equipment has entered a geofence.

NOTE

If the equipment had moved into the subsite after business hours, no event would be published and the operations manager wouldn't receive any notifications.

Location 4 (47.637988,-122.1338344)

1. Near the top of the Postman app, select **New**. In the **Create New** window, select **Request**. Enter a **Request name** for the request. Make it *Location 4*. Select the collection you created in the [Upload Geofencing GeoJSON data section](#), and then select **Save**.
2. Select the **GET** HTTP method in the builder tab, and enter the following URL. Make sure to replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key, and `{udid}` with the `udid` you saved in the [Upload Geofencing GeoJSON data section](#).

```
https://atlas.microsoft.com/spatial/geofence/json?subscription-key={subscription-key}&api-version=1.0&deviceId=device_01&udid={udid}&lat=47.637988&userTime=2023-01-16&lon=-122.1338344&searchBuffer=5&isAsync=True&mode=EnterAndExit
```

3. Select **Send**. The following GeoJSON appears in the response window:

```
{
  "geometries": [
    {
      "deviceId": "device_01",
      "udId": "64f71aa5-bbee-942d-e351-651a6679a7da",
      "geometryId": "1",
      "distance": -999.0,
      "nearestLat": 47.637985,
      "nearestLon": -122.133907
    }
  ],
  "expiredGeofenceGeometryId": [
    "2"
  ],
  "invalidPeriodGeofenceGeometryId": [],
  "isEventPublished": false
}
```

In the preceding GeoJSON response, the equipment has remained in the main site geofence, but has exited the subsite geofence. Notice, however, that the `userTime` value is after the `expiredTime` as defined in the geofence data. As a result, the `isEventPublished` parameter is set to `false`, and the operations manager doesn't receive an email notification.

Location 5 (47.63799, -122.134505)

1. Near the top of the Postman app, select **New**. In the **Create New** window, select **Request**. Enter a **Request name** for the request. Make it *Location 5*. Select the collection you created in the [Upload Geofencing GeoJSON data section](#), and then select **Save**.
2. Select the **GET** HTTP method in the builder tab, and enter the following URL. Make sure to replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key, and `{udid}` with the `udid` you saved in the [Upload Geofencing GeoJSON data section](#).

```
https://atlas.microsoft.com/spatial/geofence/json?subscription-key={subscription-key}&api-version=1.0&deviceId=device_01&udid={udid}&lat=47.63799&lon=-122.134505&searchBuffer=5&isAsync=True&mode=EnterAndExit
```

3. Select **Send**. The following GeoJSON appears in the response window:

```
{  
  "geometries": [  
    {  
      "deviceId": "device_01",  
      "udId": "64f71aa5-bbee-942d-e351-651a6679a7da",  
      "geometryId": "1",  
      "distance": -999.0,  
      "nearestLat": 47.637985,  
      "nearestLon": -122.133907  
    },  
    {  
      "deviceId": "device_01",  
      "udId": "64f71aa5-bbee-942d-e351-651a6679a7da",  
      "geometryId": "2",  
      "distance": 999.0,  
      "nearestLat": 47.637945,  
      "nearestLon": -122.133683  
    }  
  ],  
  "expiredGeofenceGeometryId": [],  
  "invalidPeriodGeofenceGeometryId": [],  
  "isEventPublished": true  
}
```

In the preceding GeoJSON response, the equipment has exited the main site geofence. As a result, the `isEventPublished` parameter is set to `true`, and the operations manager receives an email notification indicating that the equipment has exited a geofence.

You can also [Send email notifications using Event Grid and Logic Apps](#) and check [Supported Events Handlers in Event Grid](#) using Azure Maps.

Clean up resources

There are no resources that require cleanup.

Next steps

[Handle content types in Azure Logic Apps](#)

Tutorial: Implement IoT spatial analytics by using Azure Maps

6/1/2021 • 11 minutes to read • [Edit Online](#)

In an IoT scenario, it's common to capture and track relevant events that occur in space and time. Examples include fleet management, asset tracking, mobility, and smart city applications. This tutorial guides you through a solution that tracks used car rental movement by using the Azure Maps APIs.

In this tutorial you will:

- Create an Azure storage account to log car tracking data.
- Upload a geofence to the Azure Maps Data service by using the Data Upload API.
- Create a hub in Azure IoT Hub, and register a device.
- Create a function in Azure Functions, implementing business logic based on Azure Maps spatial analytics.
- Subscribe to IoT device telemetry events from the Azure function via Azure Event Grid.
- Filter the telemetry events by using IoT Hub message routing.

Prerequisites

1. Sign in to the [Azure portal](#).
2. [Create an Azure Maps account](#).
3. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information, see [manage authentication in Azure Maps](#).
4. [Create a resource group](#). In this tutorial, we'll name our resource group *ContosoRental*, but you can choose whatever name you like.
5. Download the [rentalCarSimulation C# project](#).

This tutorial uses the [Postman](#) application, but you can choose a different API development environment.

Use case: rental car tracking

Let's say that a car rental company wants to log location information, distance traveled, and running state for its rental cars. The company also wants to store this information whenever a car leaves the correct authorized geographic region.

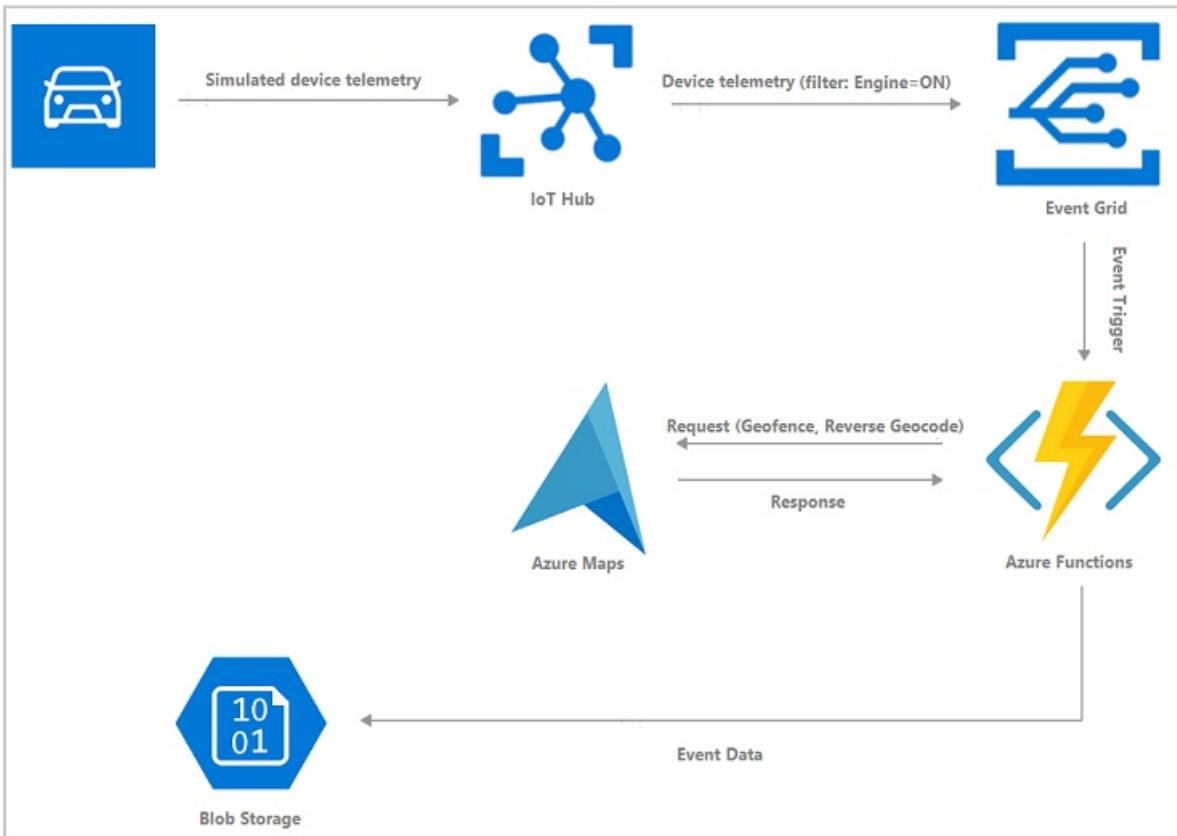
The rental cars are equipped with IoT devices that regularly send telemetry data to IoT Hub. The telemetry includes the current location and indicates whether the car's engine is running. The device location schema adheres to the IoT [Plug and Play schema for geospatial data](#). The rental car's device telemetry schema looks like the following JSON code:

```
{
  "data": {
    "properties": {
      "Engine": "ON"
    },
    "systemProperties": {
      "iothub-content-type": "application/json",
      "iothub-content-encoding": "utf-8",
      "iothub-connection-device-id": "ContosoRentalDevice",
      "iothub-connection-auth-method": ""
    },
    "IoT Hub": {
      "scope": "\device\",
      "type": "sas",
      "issuer": "iothub",
      "acceptingIpFilterRule": ":null",
      "iothub-connection-auth-generation-id": "636959817064335548",
      "iothub-enqueuedtime": "2019-06-18T00:17:20.608Z",
      "iothub-message-source": "Telemetry"
    },
    "body": {
      "location": {
        "type": "Point",
        "coordinates": [ -77.025988698005662, 38.9015330523316 ]
      }
    }
  }
}
```

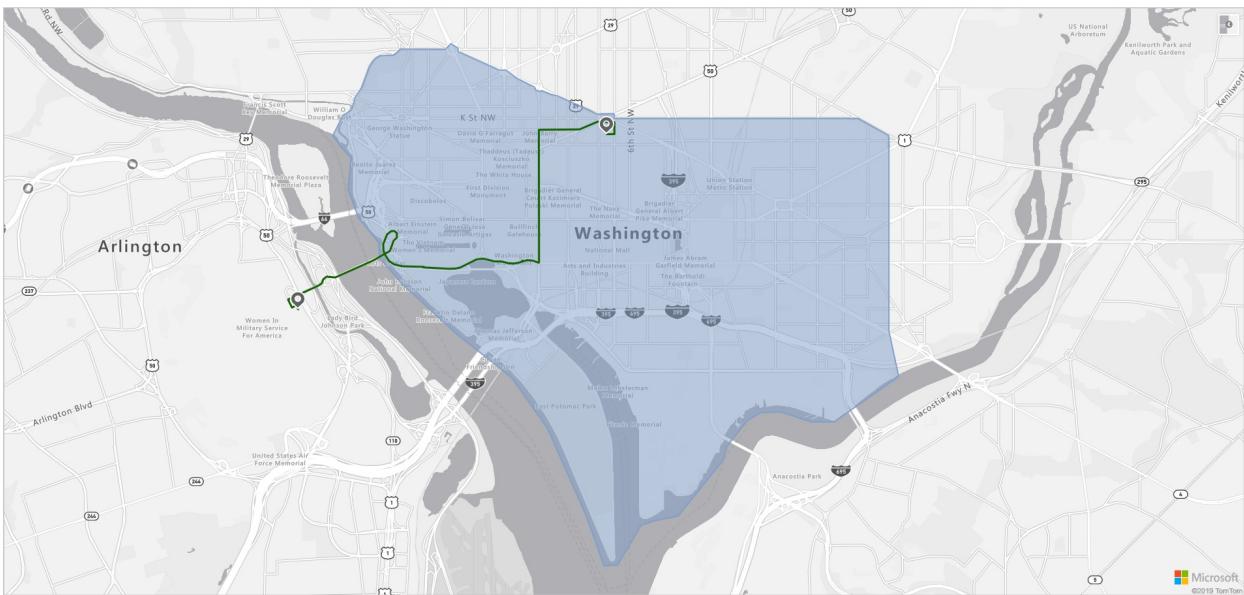
In this tutorial, you only track one vehicle. After you set up the Azure services, you need to download the [rentalCarSimulation C# project](#) to run the vehicle simulator. The entire process, from event to function execution, is summarized in the following steps:

1. The in-vehicle device sends telemetry data to IoT Hub.
2. If the car engine is running, the hub publishes the telemetry data to Event Grid.
3. An Azure function is triggered because of its event subscription to device telemetry events.
4. The function logs the vehicle device location coordinates, event time, and the device ID. It then uses the [Spatial Geofence Get API](#) to determine whether the car has driven outside the geofence. If it has traveled outside the geofence boundaries, the function stores the location data received from the event into a blob container. The function also queries the [Search Address Reverse](#) to translate the coordinate location to a street address, and stores it with the rest of the device location data.

The following diagram shows a high-level overview of the system.



The following figure highlights the geofence area in blue. The rental car's route is indicated by a green line.



Create an Azure storage account

To store car violation tracking data, create a [general-purpose v2 storage account](#) in your resource group. If you haven't created a resource group, follow the directions in [create a resource group](#). In this tutorial, you'll name your resource group *ContosoRental*.

To create a storage account, follow the instructions in [create a storage account](#). In this tutorial, name the storage account *contosorentalstorage*, but in general you can name it anything you like.

When you successfully create your storage account, you then need to create a container to store logging data.

1. Go to your newly created storage account. In the **Essentials** section, select the **Containers** link.

Home >

contosorentalstorage ✎

Storage account

Search (Ctrl+I) Open in Explorer Move Refresh Delete Feedback

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Data transfer Events Storage Explorer (preview)

Settings Access keys Geo-replication CORS Configuration Encryption Shared access signature Firewalls and virtual networks

Essentials

Resource group (change) ContosoRental

Status Primary: Available, Secondary: Available

Location Central US, East US 2

Subscription (change) ContosoSubscription

Subscription ID xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx

Tags (change) Click here to add tags

Containers Scalable, cost-effective storage for unstructured data Learn more

File shares Serverless SMB file shares Learn more

Tables Tabular data storage

Queues Effectively scale apps according to

A screenshot of the Azure Storage account overview page for 'contosorentalstorage'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Data transfer, Events, and Storage Explorer (preview). Below that is a Settings section with links for Access keys, Geo-replication, CORS, Configuration, Encryption, Shared access signature, and Firewalls and virtual networks. The main content area is titled 'Essentials' and shows details about the resource group (ContosoRental), status (Primary: Available, Secondary: Available), location (Central US, East US 2), subscription (ContosoSubscription), and subscription ID. It also includes a 'Tags' section with a link to add tags. Below this are four service cards: 'Containers' (Scalable, cost-effective storage for unstructured data, Learn more), 'File shares' (Serverless SMB file shares, Learn more), 'Tables' (Tabular data storage), and 'Queues' (Effectively scale apps according to). A red box highlights the 'Containers' card.

2. In the upper-left corner, select + Container. A panel appears on the right side of the browser. Name your container *contoso-rental-logs*, and select Create.

New container

Restore containers Refresh

Last modified Public

Name * contoso-rental-logs

Public access level Private (no anonymous access)

Advanced

Create Discard

A screenshot of the 'New container' creation dialog. At the top is the title 'New container' with a close button. Below it is a 'Name' field containing 'contoso-rental-logs' with a green checkmark. Underneath is a 'Public access level' dropdown set to 'Private (no anonymous access)'. There is also an 'Advanced' section with a dropdown arrow. At the bottom are two buttons: 'Create' in blue and 'Discard' in grey.

3. Go to the **Access keys** pane in your storage account, and copy the **Storage account name** and the **Key** value in the **key1** section. You need both of these values in the "Create an Azure Function and add an Event Grid subscription" section.

The screenshot shows the 'Access keys' section of the Azure Storage Account settings. It displays two sets of keys: 'key1' and 'key2'. Each key has a 'Key' field containing a long hexagonal string and a 'Connection string' field below it. The storage account name 'contosorentaldata' is also visible.

Upload a geofence

Next, use the [Postman app](#) to [upload the geofence](#) to Azure Maps. The geofence defines the authorized geographical area for our rental vehicle. You'll be using the geofence in your Azure function to determine whether a car has moved outside the geofence area.

Follow these steps to upload the geofence by using the Azure Maps Data Upload API:

1. Open the Postman app, and select **New**. In the **Create New** window, select **Collection**. Name the collection and select **Create**.
2. To create the request, select **New** again. In the **Create New** window, select **Request**, and enter a request name for the request. Select the collection you created in the previous step, and then select **Save**.
3. Select the **POST** HTTP method in the builder tab, and enter the following URL to upload the geofence to the Data Upload API. Make sure to replace `{subscription-key}` with your primary subscription key.

```
https://us.atlas.microsoft.com/mapData?subscription-key={subscription-key}&api-version=2.0&dataFormat=geojson
```

In the URL path, the `geojson` value against the `dataFormat` parameter represents the format of the data being uploaded.

4. Select **Body > raw** for the input format, and choose **JSON** from the drop-down list. [Open the JSON data file](#), and copy the JSON into the body section. Select **Send**.
5. Select **Send** and wait for the request to process. After the request completes, go to the **Headers** tab of the response. Copy the value of the **Operation-Location** key, which is the `status URL`.

```
https://us.atlas.microsoft.com/mapData/operations/<operationId>?api-version=2.0
```

6. To check the status of the API call, create a **GET** HTTP request on the `status URL`. You'll need to append your primary subscription key to the URL for authentication. The **GET** request should like the following URL:

```
https://us.atlas.microsoft.com/mapData/<operationId>/status?api-version=2.0&subscription-key={subscription-key}
```

7. When the request completes successfully, select the **Headers** tab in the response window. Copy the value of the **Resource-Location** key, which is the `resource location URL`. The `resource location URL` contains the unique identifier (`uid`) of the uploaded data. Copy the `uid` for later use in this tutorial.

Headers (9)		Test Results	Status: 200 OK	Time: 395 ms	Size: 746 B	Save Response ▾
KEY	VALUE					
Content-Length ⓘ	243					
Content-Type ⓘ	application/json; charset=utf-8					
x-ms-azurermaps-reg... ⓘ	West US 2					
X-Content-Type-Opti... ⓘ	nosniff					
Strict-Transport-Sec... ⓘ	max-age=31536000; includeSubDomains					
Resource-Location ⓘ	https://us.atlas.microsoft.com/mapData/metadata/6ebf1ae1-2a66-760b-e28c-b9381fcff335?api-version=2.0					
X-Cache ⓘ	CONFIG_NOCACHE					
X-MSEdge-Ref ⓘ	Ref A: 145D7FD71F604535B6952C25E019A227 Ref B: WSTEDGE1120 Ref C: 2021-05-20T16:38:20Z					
Date ⓘ	Thu, 20 May 2021 16:38:20 GMT					

Create an IoT hub

IoT Hub enables secure and reliable bi-directional communication between an IoT application and the devices it manages. For this tutorial, you want to get information from your in-vehicle device to determine the location of the rental car. In this section, you create an IoT hub within the *ContosoRental* resource group. This hub will be responsible for publishing your device telemetry events.

NOTE

The ability to publish device telemetry events on Event Grid is currently in preview. This feature is available in all regions except the following: East US, West US, West Europe, Azure Government, Azure China 21Vianet, and Azure Germany.

To create an IoT hub in the *ContosoRental* resource group, follow the steps in [create an IoT hub](#).

Register a device in your IoT hub

Devices can't connect to the IoT hub unless they're registered in the IoT hub identity registry. Here, you'll create a single device with the name, *InVehicleDevice*. To create and register the device within your IoT hub, follow the steps in [register a new device in the IoT hub](#). Make sure to copy the primary connection string of your device. You'll need it later.

Create a function and add an Event Grid subscription

Azure Functions is a serverless compute service that allows you to run small pieces of code ("functions"), without the need to explicitly provision or manage compute infrastructure. To learn more, see [Azure Functions](#).

A function is triggered by a certain event. Here, you'll create a function that is triggered by an Event Grid trigger. Create the relationship between trigger and function by creating an event subscription for IoT Hub device telemetry events. When a device telemetry event occurs, your function is called as an endpoint, and receives the relevant data for the device you previously registered in IoT Hub.

Here's the [C# script code that your function will contain](#).

Now, set up your Azure function.

1. In the Azure portal dashboard, select **Create a resource**. Type **Function App** in the search text box. Select **Function App > Create**.
2. On the **Function App** creation page, name your function app. Under **Resource Group**, select **ContosoRental** from the drop-down list. Select **.NET Core** as the **Runtime Stack**. At the bottom of the page, select **Next: Hosting >**.

Create Function App

Basics

Hosting

Monitoring

Tags

Review + create

Create a function app, which lets you group functions as a logical unit for easier management, deployment and sharing of resources. Functions lets you execute your code in a serverless environment without having to first create a VM or publish a web application.

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

ContosoSubscription

Resource Group * ⓘ

ContosoRental

[Create new](#)

Instance Details

Function App name *

ContosoRental-Functions

.azurewebsites.net

Publish *

Code Docker Container

Runtime stack *

.NET Core

Version *

3.1

Region *

Central US

[Review + create](#)

< Previous

Next : Hosting >

3. For **Storage account**, select the storage account you created in [Create an Azure storage account](#). Select **Review + create**.
4. Review the function app details, and select **Create**.
5. After the app is created, you add a function to it. Go to the function app. Select the **Functions** pane. At the top of the page, select **+ Add**. The function template panel appears. Scroll down the panel, and select **Azure Event Grid trigger**.

IMPORTANT

The **Azure Event Hub Trigger** and the **Azure Event Grid Trigger** templates have similar names. Make sure you select the **Azure Event Grid Trigger** template.

The screenshot shows the Azure Functions blade for the 'ContosoRental-Functions' function app. The left sidebar has 'Functions' selected. The main area shows a list of available triggers and activities. A red box highlights the 'Add' button and the 'Azure Event Grid trigger' card.

6. Give the function a name. In this tutorial, you'll use the name, *GetGeoFunction*, but in general you can use any name you like. Select **Create function**.
7. In the left menu, select the **Code + Test** pane. Copy and paste the **C# script** into the code window.

The screenshot shows the 'Code + Test' pane for the 'GetGeoFunction' function. The 'Code + Test' tab is selected. The code editor contains the following C# script:

```

1  #r "Microsoft.Azure.EventGrid"
2  #r "Microsoft.WindowsAzure.Storage"
3  #r "Newtonsoft.Json"
4
5  using System.Net;
6  using System.Linq;
7  using System.Text;
8  using Newtonsoft.Json;
9  using Microsoft.AspNetCore.Mvc;
10 using Microsoft.Extensions.Logging;
11 using Microsoft.WindowsAzure.Storage;
12 using Microsoft.Azure.EventGrid.Models;
13 using Microsoft.WindowsAzure.Storage.Blob;
14
15 const string SUBSCRIPTION_KEY = "<Your Azure Maps subscription key>";
16 const string UDID = "<UDID>";
17 const string STORAGE_ACCESS_KEY = "<Storage Access key>";
18 const string STORAGE_ACCOUNT_NAME = "<Storage Account name>";
19 const string STORAGE_CONTAINER_NAME = "Storage data container name";
20
21 public static async Task Run(EventGridEvent eventGridEvent, ILogger log){
22
23     // Parse event data
24     string eventData = eventGridEvent.Data.ToString();
25     var client = new HttpClient();
26
27     await CreateBlobAsync(eventData, client, SUBSCRIPTION_KEY, log);
}
    
```

8. In the C# code, replace the following parameters:
 - Replace **SUBSCRIPTION_KEY** with your Azure Maps account primary subscription key.
 - Replace **UDID** with the **udid** of the geofence you uploaded in [Upload a geofence](#).
 - The `CreateBlobAsync` function in the script creates a blob per event in the data storage account. Replace the **ACCESS_KEY**, **ACCOUNT_NAME**, and **STORAGE_CONTAINER_NAME** with your

storage account's access key, account name, and data storage container. These values were generated when you created your storage account in [Create an Azure storage account](#).

9. In the left menu, select the **Integration** pane. Select **Event Grid Trigger** in the diagram. Type in a name for the trigger, *eventGridEvent*, and select **Create Event Grid subscription**.

The screenshot shows the Azure Functions blade for the 'GetGeoFunction' function. On the left, under the 'Developer' tab, the 'Integration' section is selected. In the main area, the 'Trigger' section shows a single entry: 'Event Grid Trigger (eventGridEvent)'. This entry is highlighted with a red box. To the right, the 'Edit Trigger' pane is open, also showing the 'Event Grid Trigger' selected in the 'Binding Type' dropdown. The 'Event Trigger parameter name' field contains 'eventGridEvent'. At the bottom of this pane, a button labeled 'Create Event Grid subscription' is also highlighted with a red box.

10. Fill out the subscription details. Name the event subscription. For **Event Schema**, select **Event Grid Schema**. For **Topic Types**, select **Azure IoT Hub Accounts**. For **Resource Group**, select the resource group you created at the beginning of this tutorial. For **Resource**, select the IoT hub you created in "Create an Azure IoT hub." For **Filter to Event Types**, select **Device Telemetry**.

After choosing these options, you'll see the **Topic Type** change to **IoT Hub**. For **System Topic Name**, you can use the same name as your resource. Finally, in the **Endpoint details** section, select **Select an endpoint**. Accept all settings and select **Confirm Selection**.

Create Event Subscription

Event Grid

Basic Filters Additional Features

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

EVENT SUBSCRIPTION DETAILS

Name

Contoso-Events-Subs



Event Schema

Event Grid Schema



TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Types

Azure IoT Hub Accounts



Subscription

ContosoSubscription



Resource Group

ContosoRental



Resource

ContosoRentalHub



EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types

Device Telemetry



ENDPOINT DETAILS

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type

⚡ Azure Function (change)

Endpoint

Select an endpoint

Create

11. Review your settings. Make sure that the endpoint specifies the function you created in the beginning of this section. Select **Create**.

Create Event Subscription

Event Grid

Basic Filters Additional Features Advanced Editor

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

EVENT SUBSCRIPTION DETAILS

Name	Contoso-Events-Subs	✓
Event Schema	Event Grid Schema	▼

TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type	IoT Hub
Source Resource	ContosoRentalHub (change)
System Topic Name	

EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types	Device Telemetry	▼
-----------------------	------------------	---

ENDPOINT DETAILS

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type	Azure Function (change)
Endpoint	GetGeoFunction (change)

Create

12. Now you're back at the **Edit Trigger** panel. Select **Save**.

Filter events by using IoT Hub message routing

When you add an Event Grid subscription to the Azure function, a messaging route is automatically created in the specified IoT hub. Message routing allows you to route different data types to various endpoints. For example, you can route device telemetry messages, device life-cycle events, and device twin change events. For more information, see [Use IoT Hub message routing](#).

ContosoRentalHub | Message routing ✖

Send data from your devices to endpoints that you choose.

Routes [Custom endpoints](#) [Enrich messages](#)

Disable fallback route

[Add](#) [Test all routes](#) [Delete](#)

<input type="checkbox"/>	Name	Data Source	Routing Query	Endpoint	Enabled
<input checked="" type="checkbox"/>	RouteToEventGrid	DeviceMessages	true	eventgrid	true

Explorers

- [Query explorer](#)
- [IoT devices](#)

Automatic Device Management

- [IoT Edge](#)
- [IoT device configuration](#)

Messaging

- [File upload](#)
- [Message routing](#)

Security

The 'Message routing' item in the left sidebar is highlighted with a red box.

In your example scenario, you only want to receive messages when the rental car is moving. Create a routing query to filter the events where the `Engine` property equals "ON". To create a routing query, select the **RouteToEventGrid** route and replace the **Routing query** with "`Engine='ON'`". Then select **Save**. Now the IoT hub only publishes device telemetry where the engine is on.

Dashboard > Contoso-Rental-Hub - Message routing > RouteToEventGrid

RouteToEventGrid

Route details

We've created this default route based on your subscription to device telemetry and included your endpoint and data source information. This single route can handle all of your Event Grid subscriptions. To filter messages before telemetry data is sent, update your routing query. [Learn more](#)

Name
RouteToEventGrid

* Endpoint [?](#)
eventgrid

* Data source [?](#)
Device Telemetry Messages

* Enable route [?](#)
 Enable Disable

Create a query to filter messages before data is sent to Event Grid. Your query will be used for every message and for all of your subscriptions. [Learn more](#)

Routing query [?](#)
1 Engine='ON'

Test

Save

TIP

There are various ways to query IoT device-to-cloud messages. To learn more about message routing syntax, see [IoT Hub message routing](#).

Send telemetry data to IoT Hub

When your Azure function is running, you can now send telemetry data to the IoT hub, which will route it to Event Grid. Use a C# application to simulate location data for an in-vehicle device of a rental car. To run the application, you need the .NET Core SDK 2.1.0 or later on your development computer. Follow these steps to send simulated telemetry data to the IoT hub:

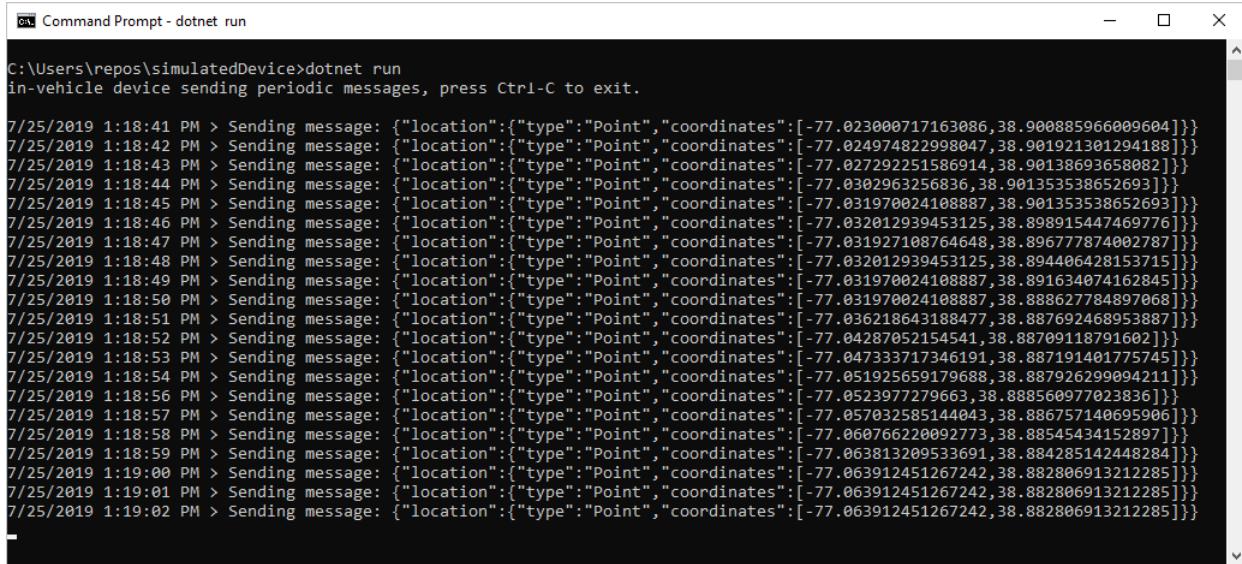
1. If you haven't done so already, download the [rentalCarSimulation](#) C# project.
2. Open the `simulatedCar.cs` file in a text editor of your choice, and replace the value of the `connectionString` with the one you saved when you registered the device. Save changes to the file.
3. Make sure you have .NET Core installed on your machine. In your local terminal window, go to the root folder of the C# project and run the following command to install the required packages for simulated device application:

```
dotnet restore
```

4. In the same terminal, run the following command to build and run the rental car simulation application:

```
dotnet run
```

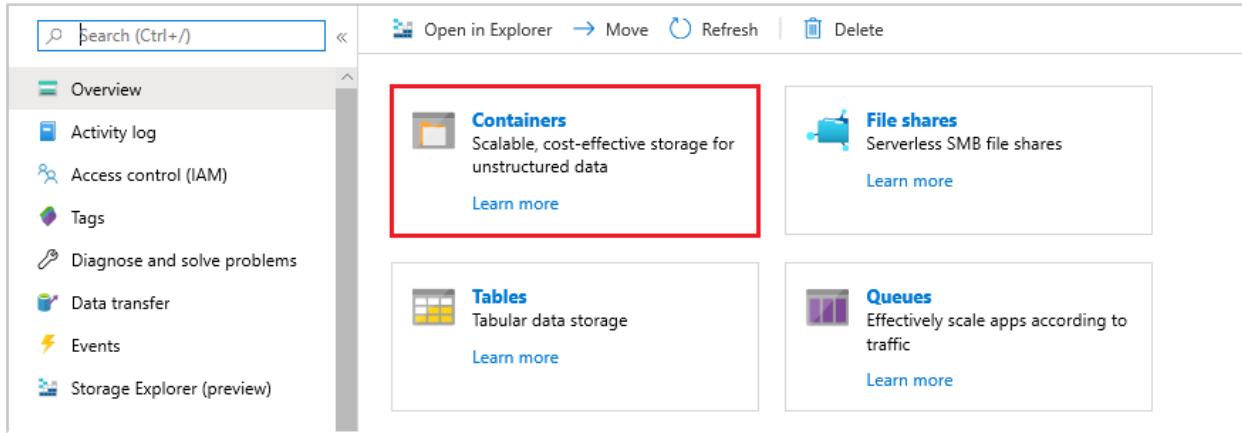
Your local terminal should look like the one below.



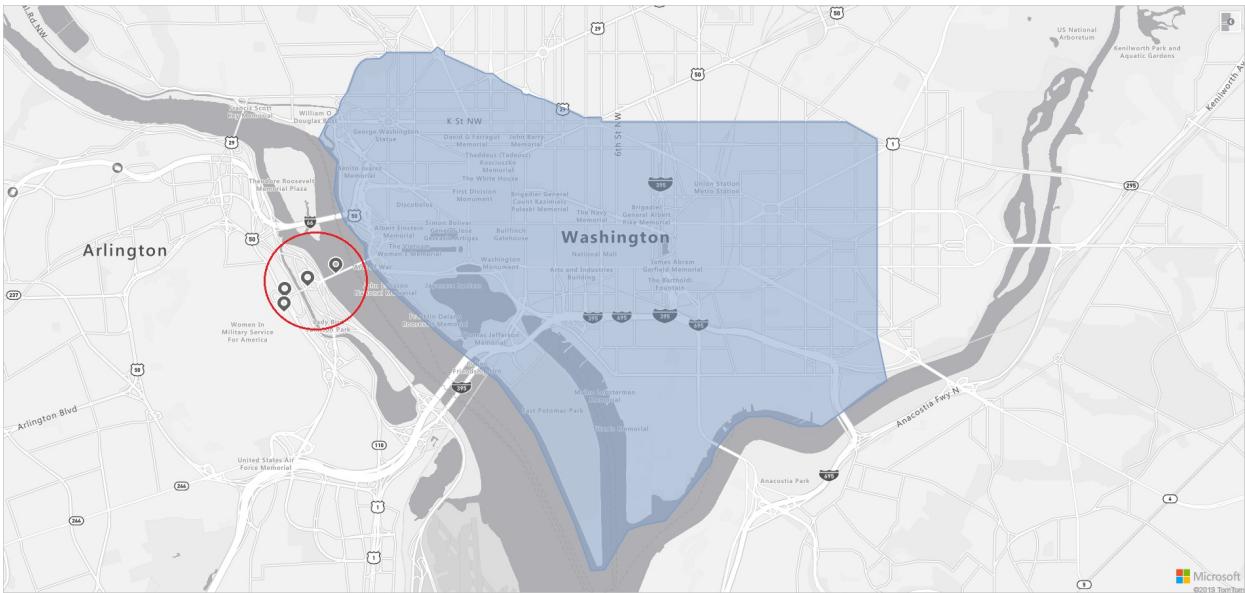
```
C:\Users\repos\simulatedDevice>dotnet run
in-vehicle device sending periodic messages, press Ctrl-C to exit.

7/25/2019 1:18:41 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.023000717163086,38.900885966009604]}}
7/25/2019 1:18:42 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.024974822998047,38.901921301294188]}}
7/25/2019 1:18:43 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.027292251586914,38.901386936580821]}}
7/25/2019 1:18:44 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.0302963256836,38.901353538652693]}}
7/25/2019 1:18:45 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.031970024108887,38.901353538652693]}}
7/25/2019 1:18:46 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.032012939453125,38.898915447469776]}}
7/25/2019 1:18:47 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.031927108764648,38.896777874002787]}}
7/25/2019 1:18:48 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.032012939453125,38.894406428153715]}}
7/25/2019 1:18:49 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.031970024108887,38.891634074162845]}}
7/25/2019 1:18:50 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.031970024108887,38.888627784897068]}}
7/25/2019 1:18:51 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.036218643188477,38.887692468953887]}}
7/25/2019 1:18:52 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.04287052154541,38.88709118791602]}}
7/25/2019 1:18:53 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.047333717346191,38.887191401775745]}}
7/25/2019 1:18:54 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.051925659179688,38.887926299094211]}}
7/25/2019 1:18:56 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.0523977279663,38.888560977023836]}}
7/25/2019 1:18:57 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.057032585144043,38.886757140695906]}}
7/25/2019 1:18:58 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.060766220092773,38.88545434152897]}}
7/25/2019 1:18:59 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.063813209533691,38.884285142448284]}}
7/25/2019 1:19:00 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.063912451267242,38.882806913212285]}}
7/25/2019 1:19:01 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.063912451267242,38.882806913212285]}}
7/25/2019 1:19:02 PM > Sending message: {"location":{"type":"Point","coordinates":[-77.063912451267242,38.882806913212285]}}
```

If you open the blob storage container now, you can see four blobs for locations where the vehicle was outside the geofence.



The following map shows four vehicle location points outside the geofence. Each location was logged at regular time intervals.



Explore Azure Maps and IoT

To explore the Azure Maps APIs used in this tutorial, see:

- [Get Search Address Reverse](#)
- [Get Geofence](#)

For a complete list of Azure Maps REST APIs, see:

- [Azure Maps REST APIs](#)
- [IoT Plug and Play](#)

To get a list of devices that are Azure certified for IoT, visit:

- [Azure certified devices](#)

Clean up resources

There are no resources that require cleanup.

Next steps

To learn more about how to send device-to-cloud telemetry, and the other way around, see:

[Send telemetry from a device](#)

Tutorial: Route electric vehicles by using Azure Notebooks (Python)

6/1/2021 • 9 minutes to read • [Edit Online](#)

Azure Maps is a portfolio of geospatial service APIs that are natively integrated into Azure. These APIs enable developers, enterprises, and ISVs to develop location-aware apps, IoT, mobility, logistics, and asset tracking solutions.

The Azure Maps REST APIs can be called from languages such as Python and R to enable geospatial data analysis and machine learning scenarios. Azure Maps offers a robust set of [routing APIs](#) that allow users to calculate routes between several data points. The calculations are based on various conditions, such as vehicle type or reachable area.

In this tutorial, you walk help a driver whose electric vehicle battery is low. The driver needs to find the closest possible charging station from the vehicle's location.

In this tutorial, you will:

- Create and run a Jupyter Notebook file on [Azure Notebooks](#) in the cloud.
- Call Azure Maps REST APIs in Python.
- Search for a reachable range based on the electric vehicle's consumption model.
- Search for electric vehicle charging stations within the reachable range, or isochrone.
- Render the reachable range boundary and charging stations on a map.
- Find and visualize a route to the closest electric vehicle charging station based on drive time.

Prerequisites

1. [Make an Azure Maps account](#), and [choose either Gen 2 or S1 pricing tier](#).
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.

For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

Create an Azure Notebooks project

To follow along with this tutorial, you need to create an Azure Notebooks project and download and run the Jupyter Notebook file. The Jupyter Notebook file contains Python code, which implements the scenario in this tutorial. To create an Azure Notebooks project and upload the Jupyter Notebook document to it, do the following steps:

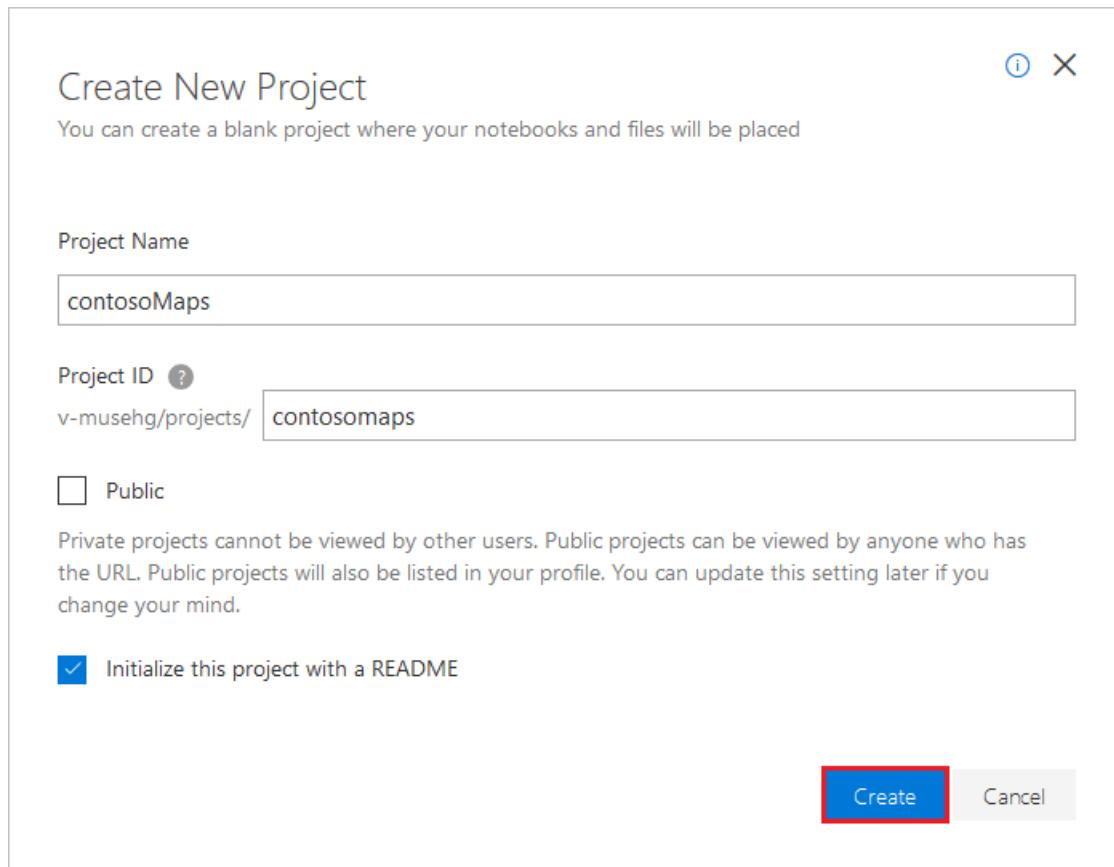
1. Go to [Azure Notebooks](#) and sign in. For more information, see [Quickstart: Sign in and set a user ID](#).
2. At the top of your public profile page, select **My Projects**.



3. On the **My Projects** page, select **New Project**.



4. In the **Create New Project** pane, enter a project name and project ID.



5. Select **Create**.
6. After your project is created, download this [Jupyter Notebook document file](#) from the [Azure Maps Jupyter Notebook repository](#).
7. In the projects list on the [My Projects](#) page, select your project, and then select **Upload** to upload the Jupyter Notebook document file.

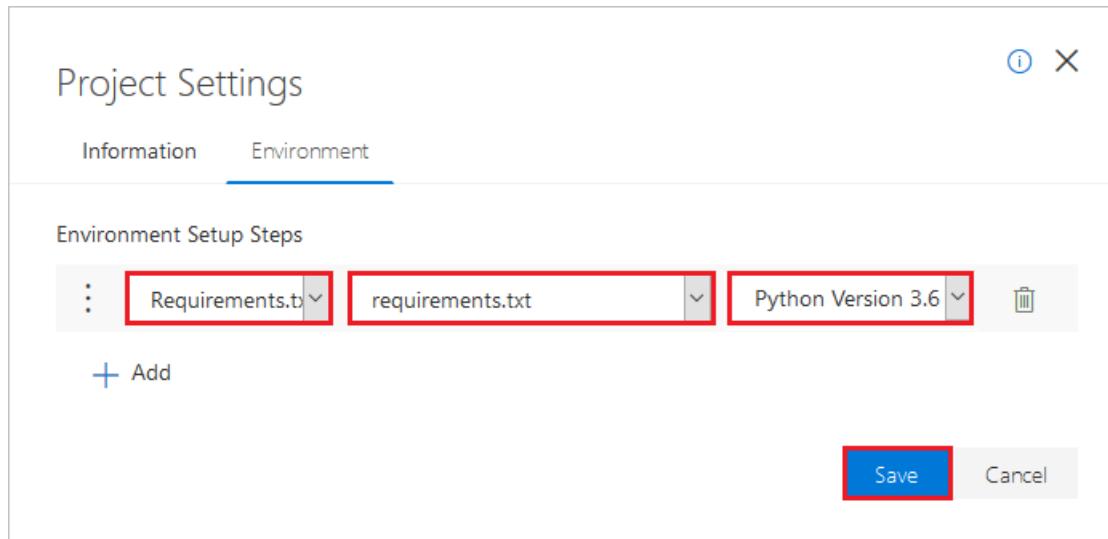
8. Upload the file from your computer, and then select **Done**.
9. After the upload has finished successfully, your file is displayed on your project page. Double-click on the file to open it as a Jupyter Notebook.

Try to understand the functionality that's implemented in the Jupyter Notebook file. Run the code, in the Jupyter Notebook file, one cell at a time. You can run the code in each cell by selecting the **Run** button at the top of the Jupyter Notebook app.

Install project level packages

To run the code in Jupyter Notebook, install packages at the project level by doing the following steps:

1. Download the [requirements.txt](#) file from the [Azure Maps Jupyter Notebook repository](#), and then upload it to your project.
2. On the project dashboard, select **Project Settings**.
3. In the **Project Settings** pane, select the **Environment** tab, and then select **Add**.
4. Under **Environment Setup Steps**, do the following:
 - a. In the first drop-down list, select **Requirements.txt**.
 - b. In the second drop-down list, select your *requirements.txt* file.
 - c. In the third drop-down list, select **Python Version 3.6** as your version.
5. Select **Save**.



Load the required modules and frameworks

To load all the required modules and frameworks, run the following script.

```
import time
import aiohttp
import urllib.parse
from IPython.display import Image, display
```

Request the reachable range boundary

A package delivery company has some electric vehicles in its fleet. During the day, electric vehicles need to be recharged without having to return to the warehouse. Every time the remaining charge drops to less than an hour, you search for a set of charging stations that are within a reachable range. Essentially, you search for a charging station when the battery is low on charge. And, you get the boundary information for that range of charging stations.

Because the company prefers to use routes that require a balance of economy and speed, the requested `routeType` is `eco`. The following script calls the [Get Route Range API](#) of the Azure Maps routing service. It uses parameters for the vehicle's consumption model. The script then parses the response to create a polygon object of the geojson format, which represents the car's maximum reachable range.

To determine the boundaries for the electric vehicle's reachable range, run the script in the following cell:

```

subscriptionKey = "Your Azure Maps key"
currentLocation = [34.028115, -118.5184279]
session = aiohttp.ClientSession()

# Parameters for the vehicle consumption model
travelMode = "car"
vehicleEngineType = "electric"
currentChargeInkWh=45
maxChargeInkWh=80
timeBudgetInSec=550
routeType="eco"
constantSpeedConsumptionInkWhPerHundredkm="50,8.2:130,21.3"

# Get boundaries for the electric vehicle's reachable range.
routeRangeResponse = await (await session.get("https://atlas.microsoft.com/route/range/json?subscription-key={}&api-version=1.0&query={}&travelMode={}&vehicleEngineType={}&currentChargeInkWh={}&maxChargeInkWh={}&timeBudgetInSec={}&routeType={}&constantSpeedConsumptionInkWhPerHundredkm={}")

.format(subscriptionKey,str(currentLocation[0]),+str(currentLocation[1]),travelMode, vehicleEngineType,
currentChargeInkWh, maxChargeInkWh, timeBudgetInSec, routeType,
constantSpeedConsumptionInkWhPerHundredkm)).json()

polyBounds = routeRangeResponse["reachableRange"]["boundary"]

for i in range(len(polyBounds)):
    coordList = list(polyBounds[i].values())
    coordList[0], coordList[1] = coordList[1], coordList[0]
    polyBounds[i] = coordList

polyBounds.pop()
polyBounds.append(polyBounds[0])

boundsData = {
    "geometry": {
        "type": "Polygon",
        "coordinates":
            [
                [
                    polyBounds
                ]
            ]
    }
}

```

Search for electric vehicle charging stations within the reachable range

After you've determined the reachable range (isochrone) for the electric vehicle, you can search for charging stations within that range.

The following script calls the Azure Maps [Post Search Inside Geometry API](#). It searches for charging stations for electric vehicle, within the boundaries of the car's maximum reachable range. Then, the script parses the response to an array of reachable locations.

To search for electric vehicle charging stations within the reachable range, run the following script:

```

# Search for electric vehicle stations within reachable range.
searchPolyResponse = await (await session.post(url = "https://atlas.microsoft.com/search/geometry/json?subscription-key={}&api-version=1.0&query=electric vehicle station&idxSet=POI&limit=50".format(subscriptionKey), json = boundsData)).json()

reachableLocations = []
for loc in range(len(searchPolyResponse["results"])):
    location = list(searchPolyResponse["results"][loc]["position"].values())
    location[0], location[1] = location[1], location[0]
    reachableLocations.append(location)

```

Upload the reachable range and charging points to Azure Maps Data service

On a map, you'll want to visualize the charging stations and the boundary for the maximum reachable range of the electric vehicle. To do so, upload the boundary data and charging stations data as geojson objects to Azure Maps Data service. Use the [Data Upload API](#).

To upload the boundary and charging point data to Azure Maps Data service, run the following two cells:

```

rangeData = {
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "properties": {},
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    polyBounds
                ]
            }
        }
    ]
}

# Upload the range data to Azure Maps Data service.
uploadRangeResponse = await session.post("https://us.atlas.microsoft.com/mapData?subscription-key={}&api-version=2.0&dataFormat=geojson".format(subscriptionKey), json = rangeData)

rangeUdidRequest = uploadRangeResponse.headers["Location"]+"&subscription-key={}".format(subscriptionKey)

while True:
    getRangeUdid = await (await session.get(rangeUdidRequest)).json()
    if 'udid' in getRangeUdid:
        break
    else:
        time.sleep(0.2)
rangeUdid = getRangeUdid["udid"]

```

```

poiData = {
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "properties": {},
            "geometry": {
                "type": "MultiPoint",
                "coordinates": reachableLocations
            }
        }
    ]
}

# Upload the electric vehicle charging station data to Azure Maps Data service.
uploadPOIsResponse = await session.post("https://us.atlas.microsoft.com/mapData?subscription-key={}&api-version=2.0&dataFormat=geojson".format(subscriptionKey), json = poiData)

poiUdidRequest = uploadPOIsResponse.headers["Location"]+"&subscription-key{}".format(subscriptionKey)

while True:
    getPoiUdid = await (await session.get(poiUdidRequest)).json()
    if 'udid' in getPoiUdid:
        break
    else:
        time.sleep(0.2)
poiUdid = getPoiUdid["udid"]

```

Render the charging stations and reachable range on a map

After you've uploaded the data to the data service, call the Azure Maps [Get Map Image service](#). This service is used to render the charging points and maximum reachable boundary on the static map image by running the following script:

```

# Get boundaries for the bounding box.
def getBounds(polyBounds):
    maxLon = max(map(lambda x: x[0], polyBounds))
    minLon = min(map(lambda x: x[0], polyBounds))

    maxLat = max(map(lambda x: x[1], polyBounds))
    minLat = min(map(lambda x: x[1], polyBounds))

    # Buffer the bounding box by 10 percent to account for the pixel size of pins at the ends of the route.
    lonBuffer = (maxLon-minLon)*0.1
    minLon -= lonBuffer
    maxLon += lonBuffer

    latBuffer = (maxLat-minLat)*0.1
    minLat -= latBuffer
    maxLat += latBuffer

    return [minLon, maxLon, minLat, maxLat]

minLon, maxLon, minLat, maxLat = getBounds(polyBounds)

path = "lcff3333|lw3|la0.80|fa0.35|||udid-{}".format(rangeUdid)
pins = "custom|an15 53||udid-{}||https://raw.githubusercontent.com/Azure-Samples/AzureMapsCodeSamples/master/AzureMapsCodeSamples/Common/images/icons/ev_pin.png".format(poiUdid)

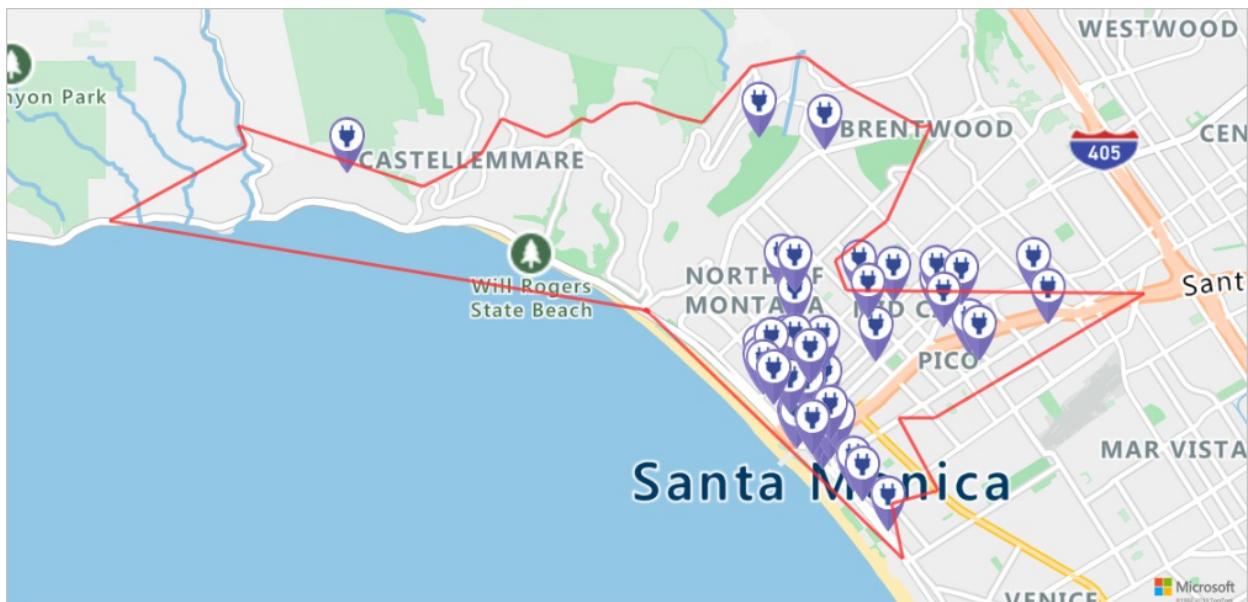
encodedPins = urllib.parse.quote(pins, safe='')

# Render the range and electric vehicle charging points on the map.
staticMapResponse = await session.get("https://atlas.microsoft.com/map/static/png?api-version=1.0&subscription-key={}&pins={}&path={}&bbox={}&zoom=12".format(subscriptionKey,encodedPins,path,str(minLon)+", "+str(minLat)+", "+str(maxLon)+", "+str(maxLat)))

poiRangeMap = await staticMapResponse.content.read()

display(Image(poiRangeMap))

```



Find the optimal charging station

First, you want to determine all the potential charging stations within the reachable range. Then, you want to know which of them can be reached in a minimum amount of time.

The following script calls the Azure Maps [Matrix Routing API](#). It returns the specified vehicle location, the travel time, and the distance to each charging station. The script in the next cell parses the response to locate the

closest reachable charging station with respect to time.

To find the closest reachable charging station that can be reached in the least amount of time, run the script in the following cell:

```
locationData = {
    "origins": {
        "type": "MultiPoint",
        "coordinates": [[currentLocation[1],currentLocation[0]]]
    },
    "destinations": {
        "type": "MultiPoint",
        "coordinates": reachableLocations
    }
}

# Get the travel time and distance to each specified charging station.
searchPolyRes = await (await session.post(url = "https://atlas.microsoft.com/route/matrix/json?subscription-key={}&api-version=1.0&routeType=shortest&waitForResults=true".format(subscriptionKey), json =
locationData)).json()

distances = []
for dist in range(len(reachableLocations)):
    distances.append(searchPolyRes["matrix"][0][dist]["response"]["routeSummary"]["travelTimeInSeconds"])

minDistLoc = []
minDistIndex = distances.index(min(distances))
minDistLoc.extend([reachableLocations[minDistIndex][1], reachableLocations[minDistIndex][0]])
closestChargeLoc = ",".join(str(i) for i in minDistLoc)
```

Calculate the route to the closest charging station

Now that you've found the closest charging station, you can call the [Get Route Directions API](#) to request the detailed route from the electric vehicle's current location to the charging station.

To get the route to the charging station and to parse the response to create a geojson object that represents the route, run the script in the following cell:

```
# Get the route from the electric vehicle's current location to the closest charging station.
routeResponse = await (await session.get("https://atlas.microsoft.com/route/directions/json?subscription-key={}&api-version=1.0&query={}:{}".format(subscriptionKey,
str(currentLocation[0])+", "+str(currentLocation[1]), closestChargeLoc))).json()

route = []
for loc in range(len(routeResponse["routes"][0]["legs"])[0]["points"])):
    location = list(routeResponse["routes"][0]["legs"])[0]["points"][loc].values()
    location[0], location[1] = location[1], location[0]
    route.append(location)

routeData = {
    "type": "LineString",
    "coordinates": route
}
```

Visualize the route

To help visualize the route, you first upload the route data as a geojson object to Azure Maps Data service . To do so, use the Azure Maps [Data Upload API](#). Then, call the rendering service, [Get Map Image API](#), to render the route on the map, and visualize it.

To get an image for the rendered route on the map, run the following script:

```

# Upload the route data to Azure Maps Data service .
routeUploadRequest = await session.post("https://atlas.microsoft.com/mapData?subscription-key={}&api-version=2.0&dataFormat=geojson".format(subscriptionKey), json = routeData)

udidRequestURI = routeUploadRequest.headers["Location"]+"&subscription-key={}".format(subscriptionKey)

while True:
    udidRequest = await (await session.get(udidRequestURI)).json()
    if 'udid' in udidRequest:
        break
    else:
        time.sleep(0.2)

udid = udidRequest["udid"]

destination = route[-1]

destination[1], destination[0] = destination[0], destination[1]

path = "lc0f6dd9|lw6||udid-{}".format(udid)
pins = "default|codb1818||{} {}|{}"
    {}".format(str(currentLocation[1]),str(currentLocation[0]),destination[1],destination[0])

# Get boundaries for the bounding box.
minLat, maxLat = (float(destination[0]),currentLocation[0]) if float(destination[0])<currentLocation[0] else (currentLocation[0], float(destination[0]))
minLon, maxLon = (float(destination[1]),currentLocation[1]) if float(destination[1])<currentLocation[1] else (currentLocation[1], float(destination[1]))

# Buffer the bounding box by 10 percent to account for the pixel size of pins at the ends of the route.
lonBuffer = (maxLon-minLon)*0.1
minLon -= lonBuffer
maxLon += lonBuffer

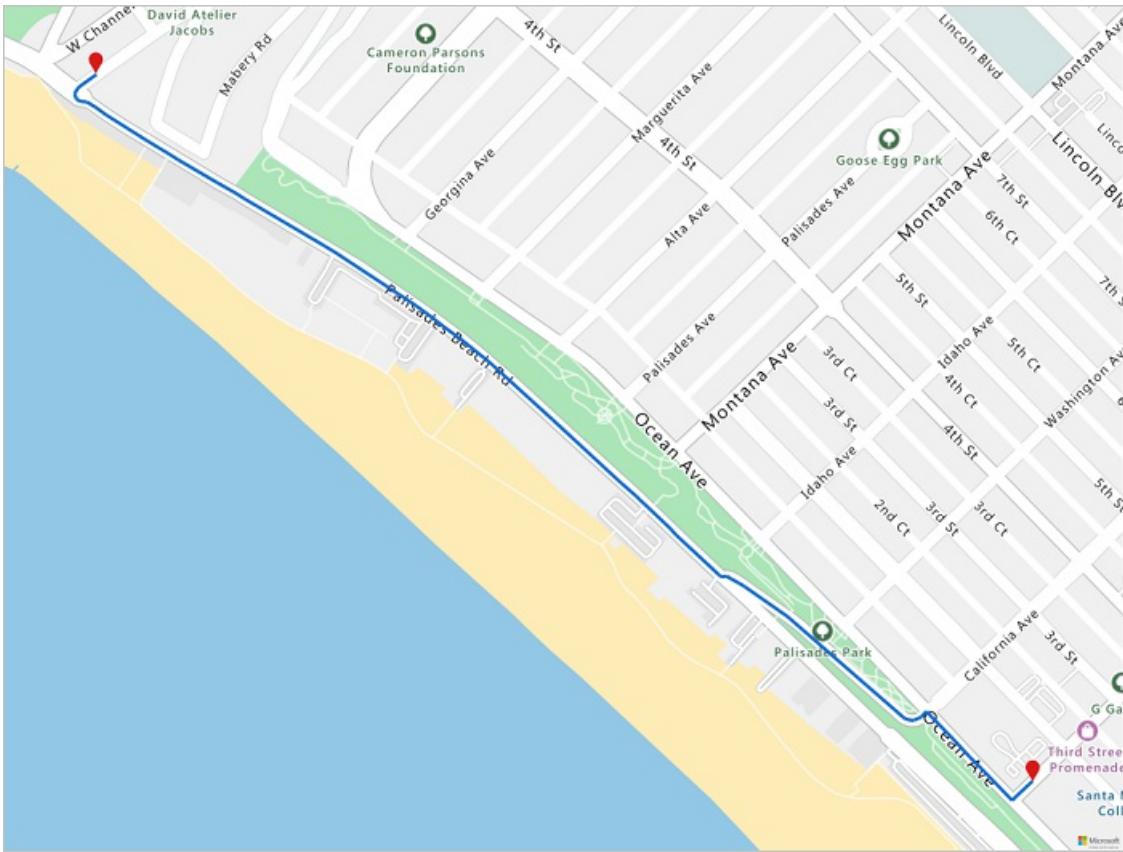
latBuffer = (maxLat-minLat)*0.1
minLat -= latBuffer
maxLat += latBuffer

# Render the route on the map.
staticMapResponse = await session.get("https://atlas.microsoft.com/map/static/png?api-version=1.0&subscription-key={}&path={}&pins={}&bbox={}&zoom=16".format(subscriptionKey,path,pins,str(minLon)+", "+str(minLat)+", "+str(maxLon)+", "+str(maxLat)))

staticMapImage = await staticMapResponse.content.read()

await session.close()
display(Image(staticMapImage))

```



In this tutorial, you learned how to call Azure Maps REST APIs directly and visualize Azure Maps data by using Python.

To explore the Azure Maps APIs that are used in this tutorial, see:

- [Get Route Range](#)
- [Post Search Inside Geometry](#)
- [Data Upload](#)
- [Render - Get Map Image](#)
- [Post Route Matrix](#)
- [Get Route Directions](#)
- [Azure Maps REST APIs](#)

Clean up resources

There are no resources that require cleanup.

Next steps

To learn more about Azure Notebooks, see

[Azure Notebooks](#)

Tutorial: Join sensor data with weather forecast data by using Azure Notebooks (Python)

4/9/2021 • 4 minutes to read • [Edit Online](#)

Wind power is one alternative energy source for fossil fuels to combat against climate change. Because wind isn't consistent by nature, wind power operators need to build machine learning (ML) models to predict the wind power capacity. This prediction is necessary to meet electricity demand and ensure the grid stability. In this tutorial, we walk through how Azure Maps weather forecast data is combined with demo data for weather readings. Weather forecast data is requested by calling Azure Maps Weather services.

In this tutorial, you will:

- Work with data files in [Azure Notebooks](#) in the cloud.
- Load demo data from file.
- Call Azure Maps REST APIs in Python.
- Render location data on the map.
- Enrich the demo data with Azure Maps [Daily Forecast](#) weather data.
- Plot forecast data in graphs.

Prerequisites

To complete this tutorial, you first need to:

1. Create an Azure Maps account subscription in the S0 pricing tier by following instructions in [Create an account](#).
2. Get the primary subscription key for your account, follow the instructions in [get primary key](#).

For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

To get familiar with Azure notebooks and to know how to get started, follow the instructions [Create an Azure Notebook](#).

NOTE

The Jupyter notebook file for this project can be downloaded from the [Weather Maps Jupyter Notebook repository](#).

Load the required modules and frameworks

To load all the required modules and frameworks, run the following script:

```
import pandas as pd
import datetime
from IPython.display import Image, display
!pip install aiohttp
import aiohttp
```

Import weather data

For the sake of this tutorial, we'll use weather data readings from sensors installed at four different wind

turbines. The sample data consists of 30 days of weather readings. These readings are gathered from weather data centers near each turbine location. The demo data contains data readings for temperature, wind speed and, direction. You can download the demo data from [here](#). The script below imports demo data to the Azure Notebook.

```
df = pd.read_csv("./data/weather_dataset_demo.csv")
```

Request daily forecast data

In our scenario, we would like to request daily forecast for each sensor location. The following script calls the [Daily Forecast API](#) of the Azure Maps Weather services. This API returns weather forecast for each wind turbine, for the next 15 days from the current date.

```
subscription_key = "Your Azure Maps key"

# Get a lists of unique station IDs and their coordinates
station_ids = pd.unique(df[['StationID']].values.ravel())
coords = pd.unique(df[['latitude','longitude']].values.ravel())

years,months,days = [],[],[]
dates_check=set()
wind_speeds, wind_direction = [], []

# Call azure maps Weather services to get daily forecast data for 15 days from current date
session = aiohttp.ClientSession()
j=-1
for i in range(0, len(coords), 2):
    wind_speeds.append([])
    wind_direction.append([])

    query = str(coords[i])+' , '+str(coords[i+1])
    forecast_response = await(await session.get("https://atlas.microsoft.com/weather/forecast/daily/json?
query={}&api-version=1.0&subscription-key={}&duration=15".format(query, subscription_key))).json()
    j+=1
    for day in range(len(forecast_response['forecasts'])):
        date = forecast_response['forecasts'][day]['date'][::10]
        wind_speeds[j].append(forecast_response['forecasts'][day]['day']['wind']['speed']['value'])
        wind_direction[j].append(forecast_response['forecasts'][day]['day']['windGust']['direction']
['degrees'])

        if date not in dates_check:
            year,month,day= date.split('-')
            years.append(year)
            months.append(month)
            days.append(day)
            dates_check.add(date)

await session.close()
```

The script below renders the turbine locations on the map by calling the Azure Maps [Get Map Image service](#).

```

# Render the turbine locations on the map by calling the Azure Maps Get Map Image service
session = aiohttp.ClientSession()

pins="default|la-25+60|ls12|lc003C62|co9B2F15||'Location A'{} {}|'Location B'{} {}|'Location C'{} {}|'Location D'{} {}".format(coords[1],coords[0],coords[3],coords[2],coords[5],coords[4],coords[7],coords[6])

image_response = "https://atlas.microsoft.com/map/static/png?subscription-key={}&api-version=1.0&layer=basic&style=main&zoom=6&center={},{}&pins={}".format(subscription_key,coords[7],coords[6],pins)

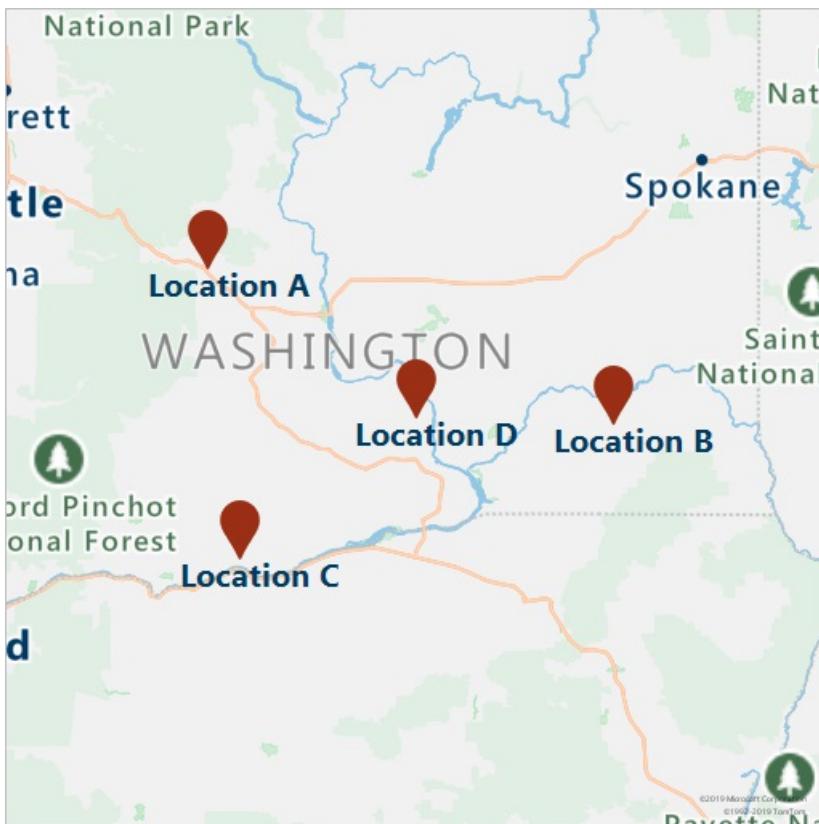
static_map_response = await session.get(image_response)

poi_range_map = await static_map_response.content.read()

await session.close()

display(Image(poi_range_map))

```



We'll group the forecast data with the demo data based on the station ID. The station ID is for the weather data center. This grouping augments the demo data with the forecast data.

```

# Group forecasted data for all locations
df = df.reset_index(drop=True)
forecast_data = pd.DataFrame(columns=['StationID','latitude','longitude','Year','Month','Day','DryBulbCelsius','WetBulbFarenheit','WetBulbCelsius','DewPointFarenheit','DewPointCelsius','RelativeHumidity','WindSpeed','WindDirection'])

for i in range(len(station_ids)):
    loc_forecast = pd.DataFrame({'StationID':station_ids[i], 'latitude':coords[0], 'longitude':coords[1], 'Year':years, 'Month':months, 'Day':days, 'WindSpeed':wind_speeds[i], 'WindDirection':wind_direction[i]})
    forecast_data = pd.concat([forecast_data,loc_forecast], axis=0, sort=False)

combined_weather_data = pd.concat([df,forecast_data])
grouped_weather_data = combined_weather_data.groupby(['StationID'])

```

The following table displays the combined historical and forecast data for one of the turbine locations.

```
# Display data for first location
grouped_weather_data.get_group(station_ids[0]).reset_index()
```

index	StationID	latitude	longitude	Year	Month	Day	DryBulbCelsius	WetBulbFarenheit	WetBulbCelsius	DewPointFarenheit	DewPointCelsius	RelativeHumidity	WindSpeed	WindDirection	
0	0	10397	47.149757	-120.805769	2019	11	9	14	56	13.2	55	13	93	6.0	230.0
1	1	10397	47.149757	-120.805769	2019	11	10	12.2	47	8.4	40	4.4	59	7.0	320.0
2	2	10397	47.149757	-120.805769	2019	11	11	10.6	40	4.5	25	-3.9	36	8.0	340.0
3	3	10397	47.149757	-120.805769	2019	11	12	8.3	46	7.5	44	6.7	89	15.0	80.0
4	4	10397	47.149757	-120.805769	2019	11	13	4.4	39	3.9	38	3.3	93	8.0	50.0
5	5	10397	47.149757	-120.805769	2019	11	14	10	44	6.4	36	2.2	59	5.0	330.0
6	6	10397	47.149757	-120.805769	2019	11	15	12.8	48	8.9	41	5	59	5.0	170.0
7	7	10397	47.149757	-120.805769	2019	11	16	16.7	55	12.7	49	9.4	63	6.0	180.0
8	8	10397	47.149757	-120.805769	2019	11	17	17.8	58	14.3	53	11.7	68	8.0	170.0
9	9	10397	47.149757	-120.805769	2019	11	18	18.9	62	16.5	59	15	78	6.0	180.0
10	10	10397	47.149757	-120.805769	2019	11	19	19.4	59	14.9	53	11.7	61	8.0	180.0
11	11	10397	47.149757	-120.805769	2019	11	20	18.3	63	16.9	61	16.1	87	18.0	190.0
12	12	10397	47.149757	-120.805769	2019	11	21	13.3	47	8.2	37	2.8	49	8.0	320.0
13	13	10397	47.149757	-120.805769	2019	11	22	16.1	49	9.3	36	2.2	39	5.0	40.0
14	14	10397	47.149757	-120.805769	2019	11	23	14	55	12.9	54	12	90	9.0	100.0
15	15	10397	47.149757	-120.805769	2019	11	24	17.2	61	15.9	59	15	87	6.0	110.0
16	16	10397	47.149757	-120.805769	2019	11	25	20.6	64	17.5	60	15.6	73	5.0	140.0
17	17	10397	47.149757	-120.805769	2019	11	26	18.9	63	17.1	61	16.1	84	7.0	180.0
18	18	10397	47.149757	-120.805769	2019	11	27	21.7	63	16.9	57	13.9	61	14.0	180.0
19	19	10397	47.149757	-120.805769	2019	11	28	6.7	40	4.4	35	1.7	71	9.0	310.0
20	20	10397	47.149757	-120.805769	2019	11	29	10.6	43	5.8	32	0	48	6.0	10.0
21	21	10397	47.149757	-120.805769	2019	11	30	12.2	47	8.4	40	4.4	59	14.0	80.0
22	22	10397	47.149757	-120.805769	2019	12	1	12.2	47	8.4	40	4.4	59	8.0	90.0
23	23	10397	47.149757	-120.805769	2019	12	2	15	54	12	49	9.4	70	7.0	200.0
24	24	10397	47.149757	-120.805769	2019	12	3	13	54	12.4	54	12	96	14.0	340.0
25	25	10397	47.149757	-120.805769	2019	12	4	11.1	43	6.3	33	0.6	49	9.0	330.0
26	26	10397	47.149757	-120.805769	2019	12	5	17.2	54	11.9	45	7.2	52	8.0	100.0
27	27	10397	47.149757	-120.805769	2019	12	6	15	57	13.7	55	13	87	7.0	350.0
28	28	10397	47.149757	-120.805769	2019	12	7	14	56	13.2	55	13	93	7.0	140.0
29	29	10397	47.149757	-120.805769	2019	12	8	16.7	60	15.6	59	15	90	6.0	80.0
30	0	10397	47.149757	-120.805769	2019	12	09	NaN	NaN	NaN	NaN	NaN	NaN	5.6	334.0
31	1	10397	47.149757	-120.805769	2019	12	10	NaN	NaN	NaN	NaN	NaN	NaN	5.6	224.0
32	2	10397	47.149757	-120.805769	2019	12	11	NaN	NaN	NaN	NaN	NaN	NaN	5.6	148.0
33	3	10397	47.149757	-120.805769	2019	12	12	NaN	NaN	NaN	NaN	NaN	NaN	7.4	292.0
34	4	10397	47.149757	-120.805769	2019	12	13	NaN	NaN	NaN	NaN	NaN	NaN	9.3	302.0
35	5	10397	47.149757	-120.805769	2019	12	14	NaN	NaN	NaN	NaN	NaN	NaN	9.3	338.0
36	6	10397	47.149757	-120.805769	2019	12	15	NaN	NaN	NaN	NaN	NaN	NaN	13.0	360.0
37	7	10397	47.149757	-120.805769	2019	12	16	NaN	NaN	NaN	NaN	NaN	NaN	1.9	156.0
38	8	10397	47.149757	-120.805769	2019	12	17	NaN	NaN	NaN	NaN	NaN	NaN	3.7	110.0
39	9	10397	47.149757	-120.805769	2019	12	18	NaN	NaN	NaN	NaN	NaN	NaN	3.7	125.0
40	10	10397	47.149757	-120.805769	2019	12	19	NaN	NaN	NaN	NaN	NaN	NaN	5.6	141.0
41	11	10397	47.149757	-120.805769	2019	12	20	NaN	NaN	NaN	NaN	NaN	NaN	3.7	119.0
42	12	10397	47.149757	-120.805769	2019	12	21	NaN	NaN	NaN	NaN	NaN	NaN	3.7	98.0
43	13	10397	47.149757	-120.805769	2019	12	22	NaN	NaN	NaN	NaN	NaN	NaN	3.7	115.0
44	14	10397	47.149757	-120.805769	2019	12	23	NaN	NaN	NaN	NaN	NaN	NaN	3.7	119.0

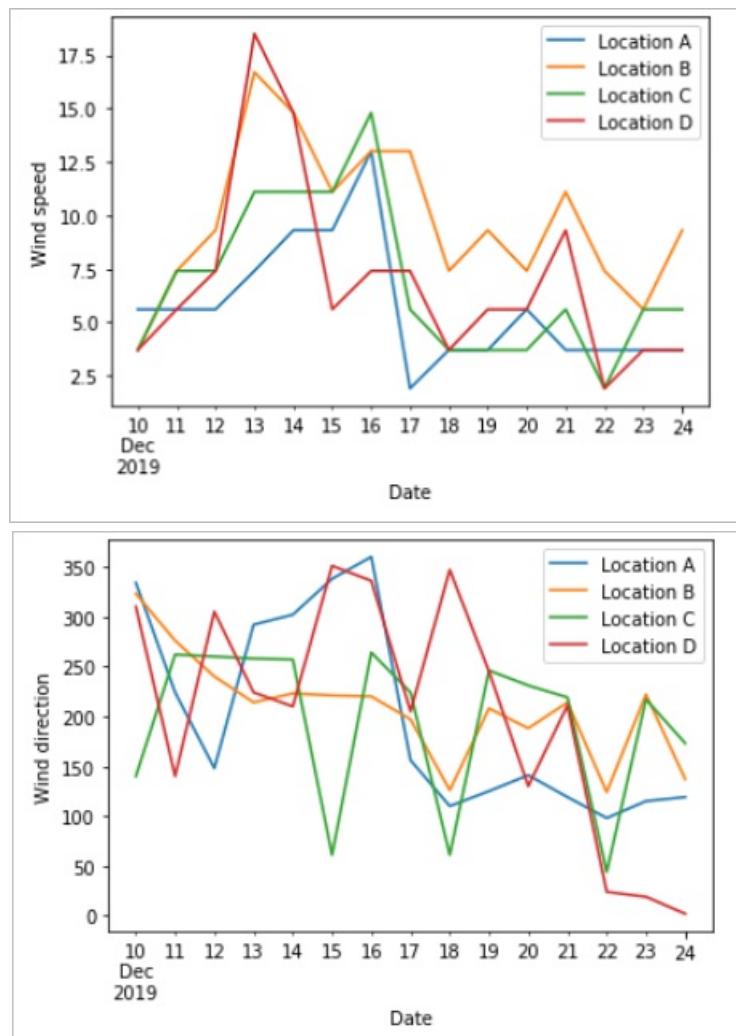
Plot forecast data

We'll plot the forecasted values against the days for which they're forecasted. This plot allows us to see the speed and direction changes of the wind for the next 15 days.

```
# Plot wind speed
curr_date = datetime.datetime.now().date()
windsPlot_df = pd.DataFrame({ 'Location A': wind_speeds[0], 'Location B': wind_speeds[1], 'Location C': wind_speeds[2], 'Location D': wind_speeds[3]}, index=pd.date_range(curr_date, periods=15))
windsPlot = windsPlot_df.plot.line()
windsPlot.set_xlabel("Date")
windsPlot.set_ylabel("Wind speed")
```

```
#Plot wind direction
windsPlot_df = pd.DataFrame({ 'Location A': wind_direction[0], 'Location B': wind_direction[1], 'Location C': wind_direction[2], 'Location D': wind_direction[3]}, index=pd.date_range(curr_date, periods=15))
windsPlot = windsPlot_df.plot.line()
windsPlot.set_xlabel("Date")
windsPlot.set_ylabel("Wind direction")
```

The graphs below visualize the forecast data. For the change of wind speed, see the left graph. For change in wind direction, see the right graph. This data is prediction for next 15 days from the day the data is requested.



In this tutorial you learned, how to call Azure Maps REST APIs to get weather forecast data. You also learned how to visualize the data on graphs.

To learn more about how to call Azure Maps REST APIs inside Azure Notebooks, see [EV routing using Azure Notebooks](#).

To explore the Azure Maps APIs that are used in this tutorial, see:

- [Daily Forecast](#)
- [Render - Get Map Image](#)

For a complete list of Azure Maps REST APIs, see [Azure Maps REST APIs](#).

Clean up resources

There are no resources that require cleanup.

Next steps

To learn more about Azure Notebooks, see

[Azure Notebooks](#)

Tutorial: Migrate from Bing Maps to Azure Maps

6/1/2021 • 4 minutes to read • [Edit Online](#)

This guide provides insights on how to migrate web, mobile and server-based applications from Bing Maps to the Azure Maps platform. This guide includes comparative code samples, migration suggestions, and best practices for migrating to Azure Maps.

In this tutorial, you'll learn:

- High-level comparison for equivalent Bing Maps features available in Azure Maps.
- What licensing differences to take into consideration.
- How to plan your migration.
- Where to find technical resources and support.

Prerequisites

1. Sign in to the [Azure portal](#). If you don't have an Azure subscription, create a [free account](#) before you begin.
2. [Make an Azure Maps account](#)
3. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

Azure Maps platform overview

Azure Maps provides developers from all industries powerful geospatial capabilities, packed with the freshest mapping data available to provide geographic context for web and mobile applications. Azure Maps is an Azure One API compliant set of REST APIs for Maps, Search, Routing, Traffic, Time Zones, Geofencing, Map Data, Weather Data, and many more services accompanied by both Web and Android SDKs to make development easy, flexible, and portable across multiple platforms. [Azure Maps is also available in Power BI](#).

High-level platform comparison

The following table provides a high-level list of Bing Maps features and the relative support for those features in Azure Maps. This list doesn't include additional Azure Maps features such as accessibility, geofencing APIs, traffic services, spatial operations, direct map tile access, and batch services.

BING MAPS FEATURE	AZURE MAPS SUPPORT
Web SDK	✓
Android SDK	✓
iOS SDK	Planned
UWP SDK	N/A
WPF SDK	N/A
REST Service APIs	✓

BING MAPS FEATURE	AZURE MAPS SUPPORT
Autosuggest	✓
Directions (including truck)	✓
Distance Matrix	✓
Elevations	✓
Imagery – Static Map	✓
Imagery Metadata	✓
Isochrones	✓
Local Insights	✓
Local Search	✓
Location Recognition	✓
Locations (forward/reverse geocoding)	✓
Optimized Itinerary Routes	Planned
Snap to roads	✓
Spatial Data Services (SDS)	Partial
Time Zone	✓
Traffic Incidents	✓
Configuration driven maps	N/A

Bing Maps provides basic key-based authentication. Azure Maps provides both basic key-based authentication and highly secure, Azure Active Directory authentication.

Licensing considerations

When migrating to Azure Maps from Bing Maps, the following information should be considered with regard to licensing.

- Azure Maps charges for the usage of interactive maps based on the number of map tiles loaded, whereas Bing Maps charges for the loading of the map control (sessions). To reduce costs for developers, Azure Maps automatically caches map tiles. One Azure Maps transaction is generated for every 15 map tiles that are loaded. The interactive Azure Maps SDKs use 512-pixel tiles, and on average generates one or less transactions per page view.
- Azure Maps allows data from its platform to be stored in Azure. It can also be cached elsewhere for up to six months as per the [terms of use](#).

Here are some licensing-related resources for Azure Maps:

- [Azure Maps pricing page](#)
- [Azure pricing calculator](#)
- [Azure Maps term of use](#) (included in the Microsoft Online Services Terms)
- [Choose the right pricing tier in Azure Maps](#)

Suggested migration plan

Here's an example of a high-level migration plan.

1. Take inventory of what Bing Maps SDKs and services your application is using and verify that Azure Maps provides alternative SDKs and services for you to migrate to.
2. Create an Azure subscription (if you don't already have one) at <https://azure.com>.
3. Create an Azure Maps account ([documentation](#)) and authentication key or Azure Active Directory ([documentation](#)).
4. Migrate your application code.
5. Test your migrated application.
6. Deploy your migrated application to production.

Create an Azure Maps account

To create an Azure Maps account and get access to the Azure Maps platform, follow these steps:

1. If you don't have an Azure subscription, create a [free account](#) before you begin.
2. Sign in to the [Azure portal](#).
3. Create an [Azure Maps account](#).
4. [Get your Azure Maps subscription key](#) or setup Azure Active Directory authentication for enhanced security.

Azure Maps technical resources

Here is a list of useful technical resources for Azure Maps.

- Overview: <https://azure.com/maps>
- Documentation: <https://aka.ms/AzureMapsDocs>
- Web SDK Code Samples: <https://aka.ms/AzureMapsSamples>
- Developer Forums: <https://aka.ms/AzureMapsForums>
- Videos: <https://aka.ms/AzureMapsVideos>
- Blog: <https://aka.ms/AzureMapsBlog>
- Azure Maps Feedback (UserVoice): <https://aka.ms/AzureMapsFeedback>

Migration support

Developers can seek migration support through the [forums](#) or through one of the many Azure support options: <https://azure.microsoft.com/support/options/>

New terminology

The following list contains common Bing Maps terms and their corresponding Azure Maps terms.

BING MAPS TERM	AZURE MAPS TERM
Aerial	Satellite or Aerial

BING MAPS TERM	AZURE MAPS TERM
Directions	May also be referred to as Routing
Entities	Geometries or Features
<code>EntityCollection</code>	Data source or Layer
<code>Geopoint</code>	Position
<code>GeoXML</code>	XML files in the Spatial IO module
Ground Overlay	Image layer
Hybrid (in reference to map type)	Satellite with roads
Infobox	Popup
Location	Position
<code>LocationRect</code>	Bounding box
Map Type	Map style
Navigation bar	Map style picker, Zoom control, Pitch control, Compass control
Pushpin	Bubble layer, Symbol layer, or HTML Marker

Clean up resources

There are no resources that require cleanup.

Next steps

Learn the details of how to migrate your Bing Maps application with these articles:

[Migrate a web app](#)

Tutorial: Migrate a web app from Bing Maps

3/5/2021 • 43 minutes to read • [Edit Online](#)

Web apps that use Bing Maps often use the Bing Maps V8 JavaScript SDK. The Azure Maps Web SDK is the suitable Azure-based SDK to migrate to. The Azure Maps Web SDK lets you customize interactive maps with your own content and imagery for display in your web or mobile applications. This control makes use of WebGL, allowing you to render large data sets with high performance. Develop with this SDK using JavaScript or TypeScript. In this tutorial, you will learn how to:

- Load a map
- Localize a map
- Add pushpins, polylines, and polygons.
- Display information in a popup or infobox
- Load and display KML and GeoJSON data
- Cluster pushpins
- Overlay a tile layer
- Show traffic data
- Add a ground overlay

If migrating an existing web application, check to see if it is using an open-source map control library such as Cesium, Leaflet, and OpenLayers. If it is and you would prefer to continue to use that library, you can connect it to the Azure Maps tile services ([road tiles](#) | [satellite tiles](#)). The links below provide details on how to use Azure Maps in some commonly used open-source map control libraries.

- [Cesium](#) - A 3D map control for the web. [Code samples](#) | [Plugin repo](#)
- [Leaflet](#) – Lightweight 2D map control for the web. [Code samples](#) | [Plugin repo](#)
- [OpenLayers](#) - A 2D map control for the web that supports projections. [Code samples](#) | [Plugin repo](#)

If developing using a JavaScript framework, one of the following open-source projects may be useful:

- [ng-azure-maps](#) - Angular 10 wrapper around Azure maps.
- [AzureMapsControl.Components](#) - An Azure Maps Blazor component.
- [Azure Maps React Component](#) - A react wrapper for the Azure Maps control.
- [Vue Azure Maps](#) - An Azure Maps component for Vue application.

Prerequisites

1. Sign in to the [Azure portal](#). If you don't have an Azure subscription, create a [free account](#) before you begin.
2. [Make an Azure Maps account](#)
3. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

Key features support

The following table lists key API features in the Bing Maps V8 JavaScript SDK and the support of a similar API in the Azure Maps Web SDK.

BING MAPS FEATURE	AZURE MAPS WEB SDK SUPPORT
Pushpins	✓
Pushpin clustering	✓
Polylines & Polygons	✓
Ground Overlays	✓
Heat maps	✓
Tile Layers	✓
KML Layer	✓
Contour layer	Samples
Data binning layer	Included in the open-source Azure Maps Gridded Data Source module
Animated tile layer	Included in the open-source Azure Maps Animation module
Drawing tools	✓
Geocoder service	✓
Directions service	✓
Distance Matrix service	✓
Spatial Data service	N/A
Satellite/Aerial imagery	✓
Birds eye imagery	N/A
Streetside imagery	N/A
GeoJSON support	✓
GeoXML support	✓ Spatial IO module
Well-Known Text support	✓
Custom map styles	Partial

Azure Maps also has many additional [open-source modules for the web SDK](#) that extend its capabilities.

Notable differences in the web SDKs

The following are some of the key differences between the Bing Maps and Azure Maps Web SDKs to be aware of:

- In addition to providing a hosted endpoint for accessing the Azure Maps Web SDK, an NPM package is also available for embedding the Web SDK into apps if preferred. For more information, see this [documentation](#) for more information. This package also includes TypeScript definitions.
- Bing Maps provides two hosted branches of their SDK; Release and Experimental. The Experimental branch may receive multiple updates a day when new development is taking place. Azure Maps only hosts a release branch, however experimental features are created as custom modules in the open-source Azure Maps code samples project. Bing Maps used to have a frozen branch as well that was updated less frequently, thus reducing the risk of breaking changes due to a release. In Azure Maps there you can use the NPM module and point to any previous minor version release.

TIP

Azure Maps publishes both minified and unminified versions of the SDK. Simple remove `.min` from the file names. The unminified version is useful when debugging issues but be sure to use the minified version in production to take advantage of the smaller file size.

- After creating an instance of the Map class in Azure Maps, your code should wait for the maps `ready` or `load` event to fire before interacting with the map. These events ensure that all the map resources have been loaded and are ready to be accessed.
- Both platforms use a similar tiling system for the base maps, however the tiles in Bing Maps are 256 pixels in dimension while the tiles in Azure Maps are 512 pixels in dimension. As such, to get the same map view in Azure Maps as Bing Maps, a zoom level used in Bing Maps needs to be subtracted by one in Azure Maps.
- Coordinates in Bing Maps are referred to as `latitude, longitude` while Azure Maps uses `longitude, latitude`. This format aligns with the standard `[x, y]` that is followed by most GIS platforms.
- Shapes in the Azure Maps Web SDK are based on the GeoJSON schema. Helper classes are exposed through the [atlas.data namespace](#). There is also the [atlas.Shape](#) class that can be used to wrap GeoJSON objects and make them easy to update and maintain in a data bindable way.
- Coordinates in Azure Maps are defined as Position objects that can be specified as a simple number array in the format `[longitude, latitude]` or `new atlas.data.Position(longitude, latitude)`.

TIP

The Position class has a static helper function for importing coordinates that are in `latitude, longitude` format. The `atlas.data.Position.fromLatLng` function can often be replace the `new Microsoft.Maps.Location` function in Bing Maps code.

- Rather than specifying styling information on each shape that is added to the map, Azure Maps separates styles from the data. Data is stored in data sources and is connected to rendering layers that Azure Maps code uses to render the data. This approach provides enhanced performance benefit. Additionally, many layers support data-driven styling where business logic can be added to layer style options that will change how individual shapes are rendered within a layer based on properties defined in the shape.
- Azure Maps provides a bunch of useful spatial math functions in the `atlas.math` namespace, however these differ from those in the Bing Maps spatial math module. The primary difference is that Azure Maps doesn't provide built-in functions for binary operations such as union and intersection, however, since Azure Maps is based on GeoJSON that is an open standard, there are many open-source libraries available. One popular option that works well with Azure Maps and provides a ton of spatial math capabilities is [turfjs](#).

See also the [Azure Maps Glossary](#) for an in-depth list of terminology associated with Azure Maps.

Web SDK side-by-side examples

The following is a collection of code samples for each platform that cover common use cases to help you migrate your web application from Bing Maps V8 JavaScript SDK to the Azure Maps Web SDK. Code samples related to web applications are provided in JavaScript; however, Azure Maps also provides TypeScript definitions as an additional option through an [NPM module](#).

Topics

- [Load a map](#)
- [Localizing the map](#)
- [Setting the map view](#)
- [Adding a pushpin](#)
- [Adding a custom pushpin](#)
- [Adding a polyline](#)
- [Adding a polygon](#)
- [Display an infobox](#)
- [Pushpin clustering](#)
- [Add a heat map](#)
- [Overlay a tile layer](#)
- [Show traffic data](#)
- [Add a ground overlay](#)
- [Add KML data to the map](#)
- [Add drawing tools](#)

Load a map

Loading a map in both SDK's follows the same set of steps;

- Add a reference to the Map SDK.
- Add a `div` tag to the body of the page that will act as a placeholder for the map.
- Create a JavaScript function that gets called when the page has loaded.
- Create an instance of the respective map class.

Some key differences

- Bing maps requires an account key to be specified in the script reference of the API or as a map option. Authentication credentials for Azure Maps are specified as options of the map class. This can be a subscription key or Azure Active Directory information.
- Bing Maps takes in a callback function in the script reference of the API that is used to call an initialization function to load the map. With Azure Maps, the `onload` event of the page should be used.
- When using an ID to reference the `div` element that the map will be rendered in, Bing Maps uses an HTML selector (i.e. `#myMap`), whereas Azure Maps only uses the ID value (i.e. `myMap`).
- Coordinates in Azure Maps are defined as Position objects that can be specified as a simple number array in the format `[longitude, latitude]`.
- The zoom level in Azure Maps is one level lower than the Bing Maps example due to the difference in tiling system sizes between the platforms.
- By default, Azure Maps doesn't add any navigation controls to the map canvas, such as zoom buttons and map style buttons. There are however controls for adding a map style picker, zoom buttons, compass or rotation control, and a pitch control.

- An event handler is added in Azure Maps to monitor the `ready` event of the map instance. This will fire when the map has finished loading the WebGL context and all resources needed. Any post load code can be added in this event handler.

The examples below show how to load a basic map such that is centered over New York at coordinates (longitude: -73.985, latitude: 40.747) and is at zoom level 12 in Bing Maps.

Before: Bing Maps

The following code is an example of how to display a Bing Map centered and zoomed over a location.

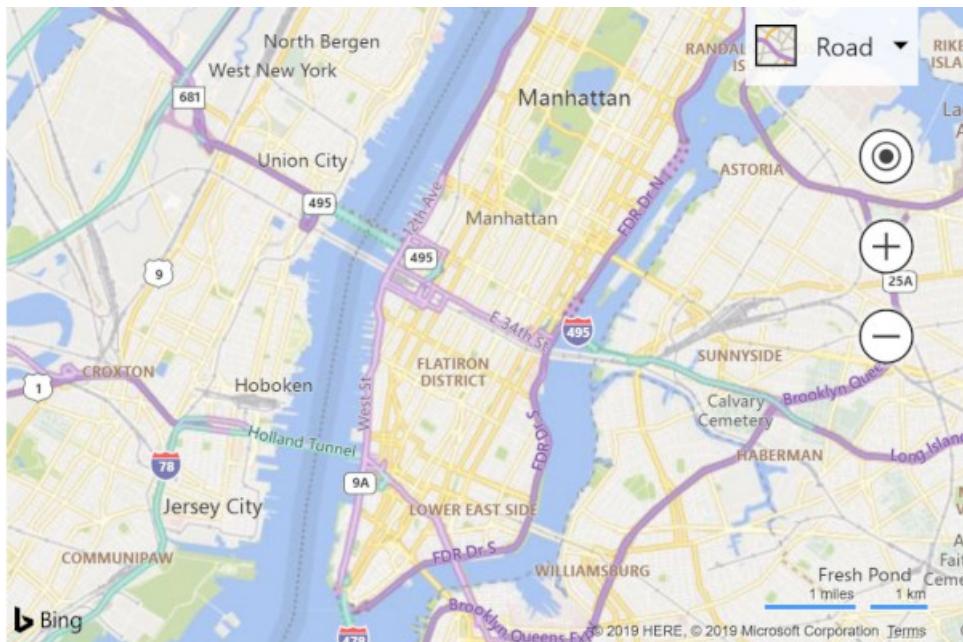
```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <script type='text/javascript'>
        var map;

        function initMap() {
            map = new Microsoft.Maps.Map('#myMap', {
                credentials: '<Your Bing Maps Key>',
                center: new Microsoft.Maps.Location(40.747, -73.985),
                zoom: 12
            });
        }
    </script>

    <!-- Bing Maps Script Reference -->
    <script src="https://www.bing.com/api/maps/mapcontrol?callback=initMap" async defer></script>
</head>
<body>
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>
```

Running this code in a browser will display a map that looks like the following image:



After: Azure Maps

The following code shows how to load a map with the same view in Azure Maps along with a map style control and zoom buttons.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css"
type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

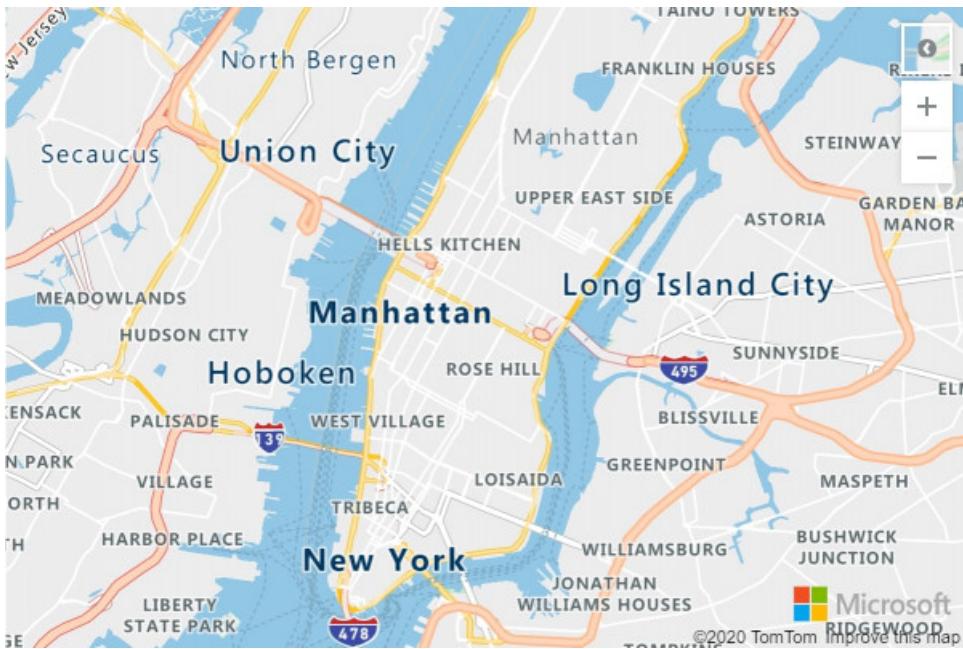
    <script type='text/javascript'>
        var map;

        function initMap() {
            map = new atlas.Map('myMap', {
                center: [-73.985, 40.747], //Format coordinates as longitude, latitude.
                zoom: 11, //Subtract the zoom level by one.

                //Add your Azure Maps key to the map SDK. Get an Azure Maps key at https://azure.com/maps.
                NOTE: The primary key should be used as the key.
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

            //Wait until the map resources are ready.
            map.events.add('ready', function () {
                //Add zoom and map style controls to top right of map.
                map.controls.add([
                    new atlas.control.StyleControl(),
                    new atlas.control.ZoomControl()
                ], {
                    position: 'top-right'
                });
            });
        }
    </script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>
```

Running this code in a browser will display a map that looks like the following image:



Detailed documentation on how to set up and use the Azure Maps map control in a web app can be found [here](#).

TIP

Azure Maps publishes both minified and unminified versions of the SDK. Remove `.min` from the file names. The unminified version is useful when debugging issues but be sure to use the minified version in production to take advantage of the smaller file size.

Additional resources

- Azure Maps also provides navigation controls for rotating and pitching the map view as documented [here](#).

Localizing the map

If your audience is spread across multiple countries or speak different languages, localization is important.

Before: Bing Maps

To localize Bing Maps, language and region are specified using the `setLang` and `UR` parameters are added to `<script>` tag reference to the API. Certain features in Bing Maps are only available in certain markets, as such the market of the user is specified using the `setMkt` parameter.

```
<script type="text/javascript" src="https://www.bing.com/api/maps/mapcontrol?callback=initMap&setLang=[language_code]&setMkt=[market]&UR=[region_code]" async defer></script>
```

Here is an example of Bing Maps with the language set to "fr-FR".



After: Azure Maps

Azure Maps only provides options for setting the language and regional view of the map. A market parameter is not used to limit features. There are two different ways of setting the language and regional view of the map. The first option is to add this information to the global `atlas` namespace that will result in all map control instances in your app defaulting to these settings. The following sets the language to French ("fr-FR") and the regional view to "Auto" :

```
atlas.setLanguage('fr-FR');
atlas.setView('auto');
```

The second option is to pass this information into the map options when loading the map like:

```
map = new atlas.Map('myMap', {
    language: 'fr-FR',
    view: 'auto',

    authOptions: {
        authType: 'subscriptionKey',
        subscriptionKey: '<Your Azure Maps Key>'
    }
});
```

NOTE

With Azure Maps it is possible to load multiple map instances on the same page with different language and region settings. Additionally, it is also possible to update these settings in the map after it has loaded. A detailed list of supported languages in Azure Maps can be found [here](#).

Here is an example of Azure Maps with the language set to "fr" and the user region set to "fr-FR".



Setting the map view

Dynamic maps in both Bing and Azure Maps can be programmatically moved to new geographic locations by calling the appropriate functions in JavaScript. The examples below show how to make the map display satellite aerial imagery, center the map over a location with coordinates (longitude: -111.0225, latitude: 35.0272) and change the zoom level to 15 in Bing Maps.

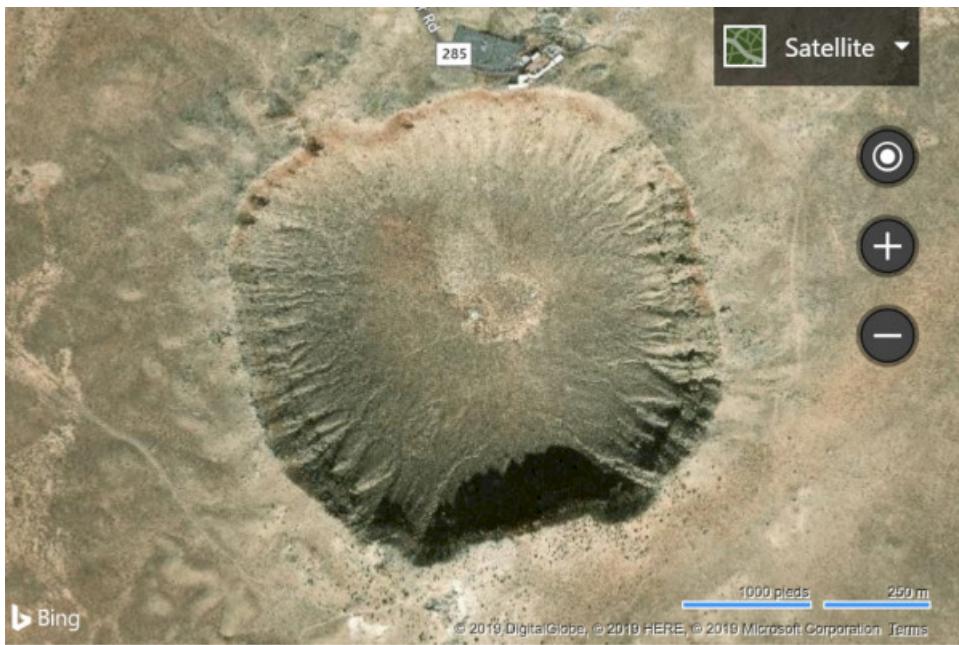
NOTE

Bing Maps uses tiles that are 256 pixels in dimensions while Azure Maps uses a larger 512-pixel tile. This reduces the number of network requests needed by Azure Maps to load the same map area as Bing Maps. However, due to the way tile pyramids work in map controls, the larger tiles in Azure Maps means that to achieve that same viewable area as a map in Bing Maps, you need to subtract the zoom level used in Bing Maps by 1 when using Azure Maps.

Before: Bing Maps

The Bing Maps map control can be programmatically moved using the `setView` function that allows you to specify the center of the map and a zoom level.

```
map.setView({
    mapTypeId: Microsoft.Maps.MapTypeId.aerial,
    center: new Microsoft.Maps.Location(35.0272, -111.0225),
    zoom: 15
});
```



After: Azure Maps

In Azure Maps, the map position can be changed programmatically by using the `setCamera` function of the map and the map style can be changed using the `setStyle` function. Note that the coordinates in Azure Maps are in "longitude, latitude" format, and the zoom level value is subtracted by 1.

```
map.setCamera({
  center: [-111.0225, 35.0272],
  zoom: 14
});

map.setStyle({
  style: 'satellite_with_roads'
});
```



Additional resources

- [Choose a map style](#)
- [Supported map styles](#)

Adding a pushpin

In Azure Maps there are multiple ways that point data can be rendered on the map;

- HTML Markers – Renders points using traditional DOM elements. HTML Markers support dragging.
- Symbol Layer – Renders points with an icon and/or text within the WebGL context.
- Bubble Layer – Renders points as circles on the map. The radii of the circles can be scaled based on properties in the data.

Both Symbol and Bubble layers are rendered within the WebGL context and are capable of rendering very large sets of points on the map. These layers require data to be stored in a data source. Data sources and rendering layers should be added to the map after the `ready` event has fired. HTML Markers are rendered as DOM elements within the page and don't use a data source. The more DOM elements a page has, the slower the page becomes. If rendering more than a few hundred points on a map, it is recommended to use one of the rendering layers instead.

The examples below add a marker to the map at (longitude: -0.2, latitude: 51.5) with the number 10 overlaid as a label.

Before: Bing Maps

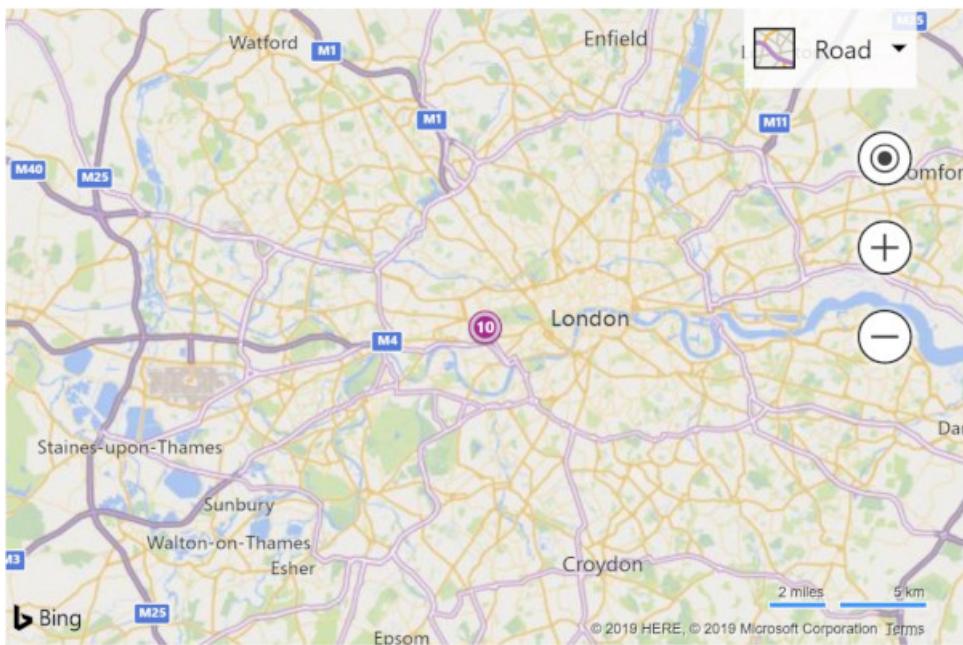
With Bing Maps, markers are added to the map using the `Microsoft.Maps.Pushpin` class*. Pushpins are then added to the map using one of two functions.

The first function is to create a layer, insert the pushpin to that and then add the layer to the map's `layers` property.

```
var pushpin = new Microsoft.Maps.Pushpin(new Microsoft.Maps.Location(51.5, -0.2), {  
    text: '10'  
});  
  
var layer = new Microsoft.Maps.Layer();  
layer.add(pushpin);  
map.layers.insert(layer);
```

The second is to add it using the map's `entities` property. This function is marked deprecated in the documentation for Bing Maps V8 however it has remained partially functional for basic scenarios.

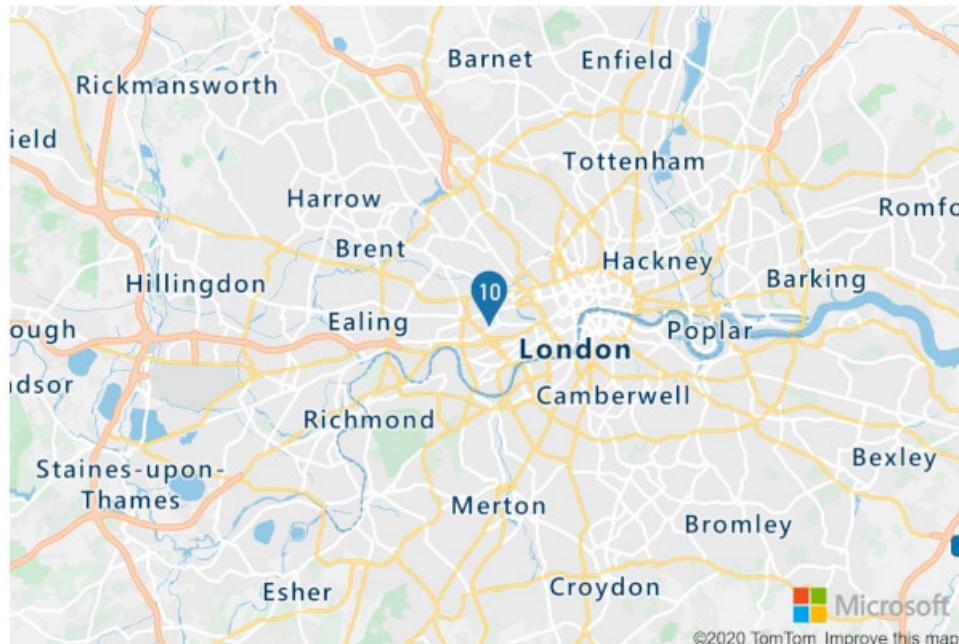
```
var pushpin = new Microsoft.Maps.Pushpin(new Microsoft.Maps.Location(51.5, -0.2), {  
    text: '10'  
});  
  
map.entities.add(pushpin);
```



After: Azure Maps using HTML Markers

In Azure Maps, HTML markers can be used to easily display a point on the map and are recommended for simple apps that only need to display a small number of points on the map. To use an HTML marker, create an instance of the `atlas.HtmlMarker` class, set the text and position options, and add the marker to the map using the `map.markers.add` function.

```
//Create a HTML marker and add it to the map.  
map.markers.add(new atlas.HtmlMarker({  
    text: '10',  
    position: [-0.2, 51.5]  
}));
```



After: Azure Maps using a Symbol Layer

When using a Symbol layer, the data must be added to a data source, and the data source attached to the layer. Additionally, the data source and layer should be added to the map after the `ready` event has fired. To render a unique text value above a symbol, the text information needs to be stored as a property of the data point and that property referenced in the `textField` option of the layer. This is a bit more work than using HTML markers but provides many performance advantages.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <script type='text/javascript'>
        var map, datasource;

        function initMap() {
            map = new atlas.Map('myMap', {
                center: [-0.2, 51.5],
                zoom: 9,

                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });
        }

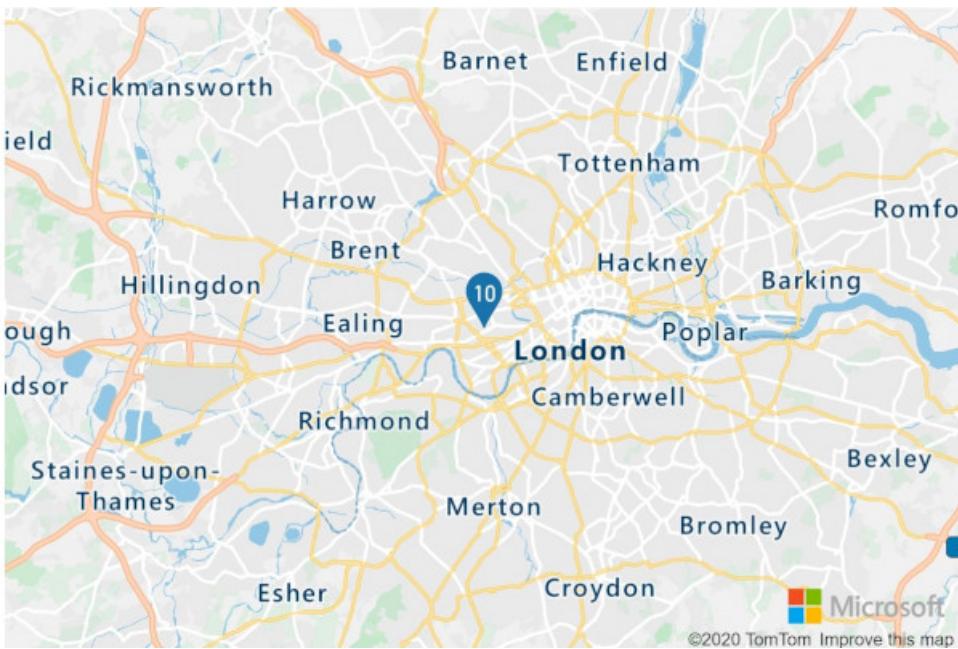
        //Wait until the map resources are ready.
        map.events.add('ready', function () {

            //Create a data source and add it to the map.
            datasource = new atlas.source.DataSource();
            map.sources.add(datasource);

            //Create a point feature, add a property to store a label for it, and add it to the data
            //source.
            datasource.add(new atlas.data.Feature(new atlas.data.Point([-0.2, 51.5]), {
                label: '10'
            }));

            //Add a layer for rendering point data as symbols.
            map.layers.add(new atlas.layer.SymbolLayer(datasource, null, {
                textOptions: {
                    //Use the label property to populate the text for the symbols.
                    textField: ['get', 'label'],
                    color: 'white',
                    offset: [0, -1]
                }
            }));
        });
    </script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



Additional resources

- [Create a data source](#)
- [Add a Symbol layer](#)
- [Add a Bubble layer](#)
- [Cluster point data](#)
- [Add HTML Markers](#)
- [Use data-driven style expressions](#)
- [Symbol layer icon options](#)
- [Symbol layer text option](#)
- [HTML marker class](#)
- [HTML marker options](#)

Adding a custom pushpin

Custom images can be used to represent points on a map. The following image is used in the below examples and uses a custom image to display a point on the map at (latitude: 51.5, longitude: -0.2) and offsets the position of the marker so that the point of the pushpin icon aligns with the correct position on the map.



yellow-pushpin.png

Before: Bing Maps

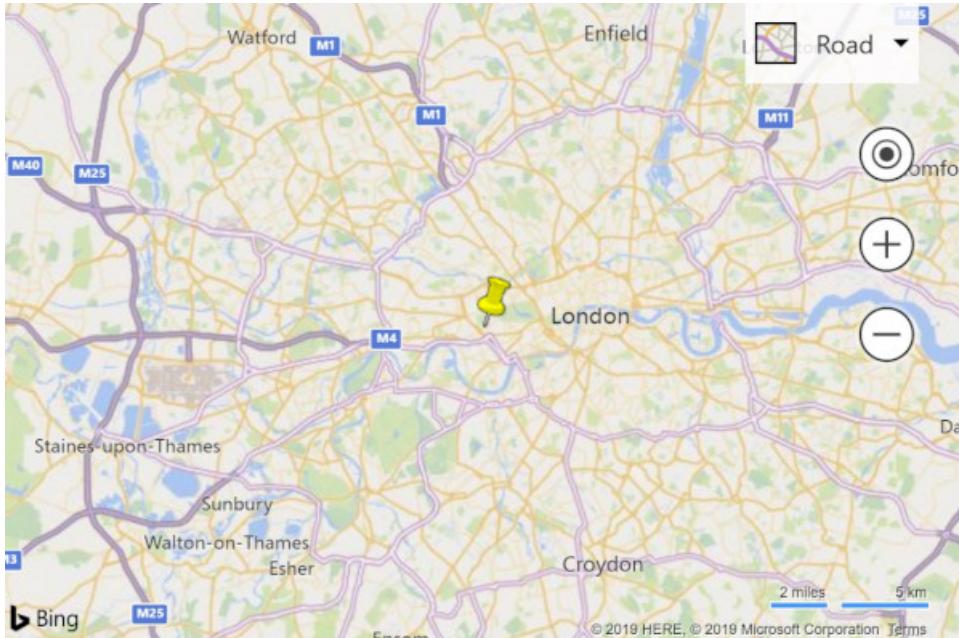
In Bing Maps, a custom marker is created by passing a URL to an image into the `icon` options of a pushpin. The `anchor` option is used to align the point of the pushpin image with the coordinate on the map. The anchor value in Bing Maps relative to the top-left corner of the image.

```

var pushpin = new Microsoft.Maps.Pushpin(new Microsoft.Maps.Location(51.5, -0.2), {
    icon: 'ylw-pushpin.png',
    anchor: new Microsoft.Maps.Point(5, 30)
});

var layer = new Microsoft.Maps.Layer();
layer.add(pushpin);
map.layers.insert(layer);

```



After: Azure Maps using HTML Markers

To customize an HTML marker in Azure Maps an HTML `string` or `HTMLElement` can be passed into the `htmlContent` option of the marker. In Azure Maps, an `anchor` option is used to specify the relative position of the marker relative to the position coordinate using one of nine defined reference points; "center", "top", "bottom", "left", "right", "top-left", "top-right", "bottom-left", "bottom-right". The content is anchored and is set to "bottom" by default, that is the bottom center of the html content. To make it easier to migrate code from Bing Maps, set the anchor to "top-left", and then use the `offset` option with the same offset used in Bing Maps. The offsets in Azure Maps move in the opposite direction of Bing Maps, so multiply them by minus one.

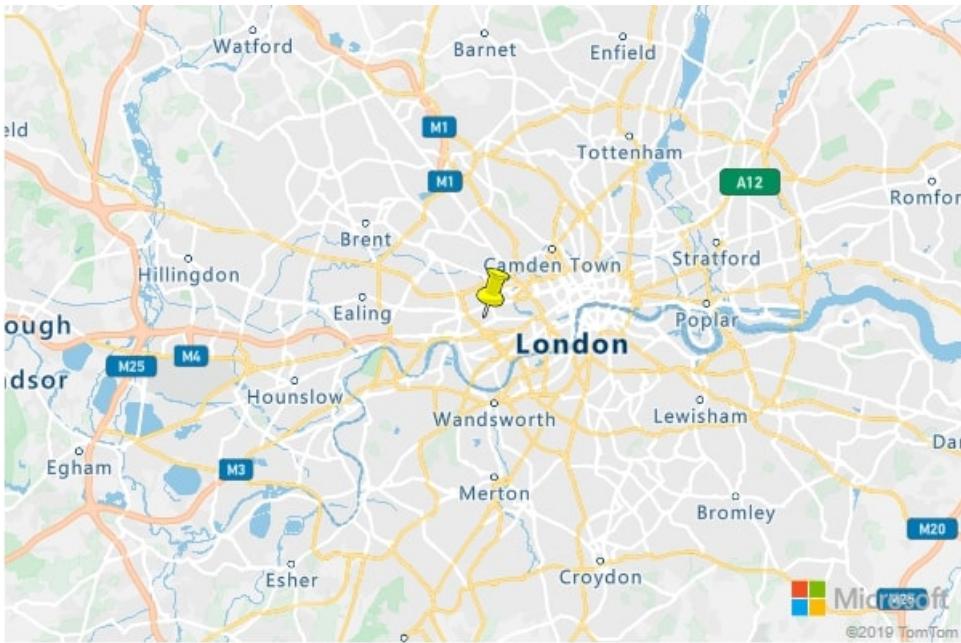
TIP

Add `pointer-events:none` as a style on the HTML content to disable the default drag behavior in MS Edge that will display an unwanted icon.

```

map.markers.add(new atlas.HtmlMarker({
    htmlContent: '',
    anchor: 'top-left',
    pixelOffset: [-5, -30],
    position: [-0.2, 51.5]
}));

```



After: Azure Maps using a Symbol Layer

Symbol layers in Azure Maps support custom images as well, but the image needs to be loaded into the map resources first and assigned a unique ID. The symbol layer can then reference this ID. The symbol can be offset to align to the correct point on the image by using the `offset` option. In Azure Maps, an `anchor` option is used to specify the relative position of the symbol relative to the position coordinate using one of nine defined reference points; "center", "top", "bottom", "left", "right", "top-left", "top-right", "bottom-left", "bottom-right". The content is anchored and set to "bottom" by default that is the bottom center of the HTML content. To make it easier to migrate code from Bing Maps, set the anchor to "top-left", and then use the `offset` option with the same offset used in Bing Maps. The offsets in Azure Maps move in the opposite direction of Bing Maps, so multiply them by minus one.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <script type='text/javascript'>
        var map, datasource;

        function initMap() {
            map = new atlas.Map('myMap', {
                center: [-0.2, 51.5],
                zoom: 9,
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

            //Wait until the map resources are ready.
            map.events.add('ready', function () {

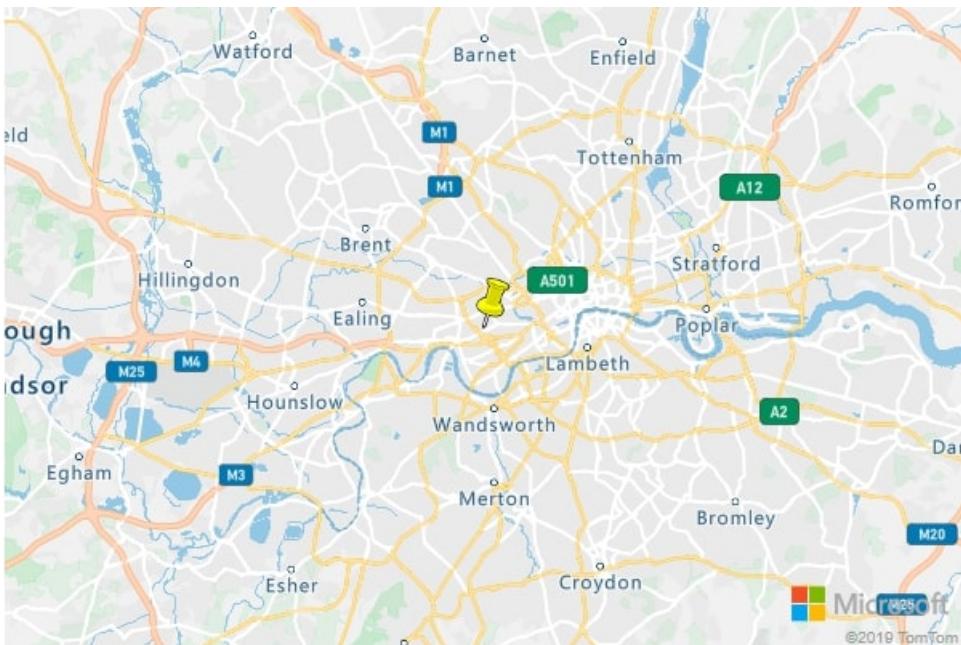
                //Load the custom image icon into the map resources.
                map.imageSprite.add('my-yellow-pin', 'ylw-pushpin.png').then(function () {

                    //Create a data source and add it to the map.
                    datasource = new atlas.source.DataSource();
                    map.sources.add(datasource);

                    //Create a point and add it to the data source.
                    datasource.add(new atlas.data.Point([-0.2, 51.5]));

                    //Add a layer for rendering point data as symbols.
                    map.layers.add(new atlas.layer.SymbolLayer(datasource, null, {
                        iconOptions: {
                            //Set the image option to the id of the custom icon that was loaded into the map
                            resources.
                            image: 'my-yellow-pin',
                            anchor: 'top-left',
                            offset: [-5, -30]
                        }
                    }));
                });
            });
        }
    </script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



TIP

To create advanced custom rendering of points, use multiple rendering layers together. For example, if you want to have multiple pushpins that have the same icon on different colored circles, instead of creating a bunch of images for each color overlay a symbol layer on top of a bubble layer and have them reference the same data source. This will be much more efficient than creating, and having the map maintain a bunch of different images.

Additional resources

- [Create a data source](#)
- [Add a Symbol layer](#)
- [Add HTML Markers](#)
- [Use data-driven style expressions](#)
- [Symbol layer icon options](#)
- [Symbol layer text option](#)
- [HTML marker class](#)
- [HTML marker options](#)

Adding a polyline

Polylines are used to represent a line or path on the map. The examples below show how to create a dashed polyline on the map.

Before: Bing Maps

In Bing Maps, the Polyline class takes in an array of locations and a set of options.

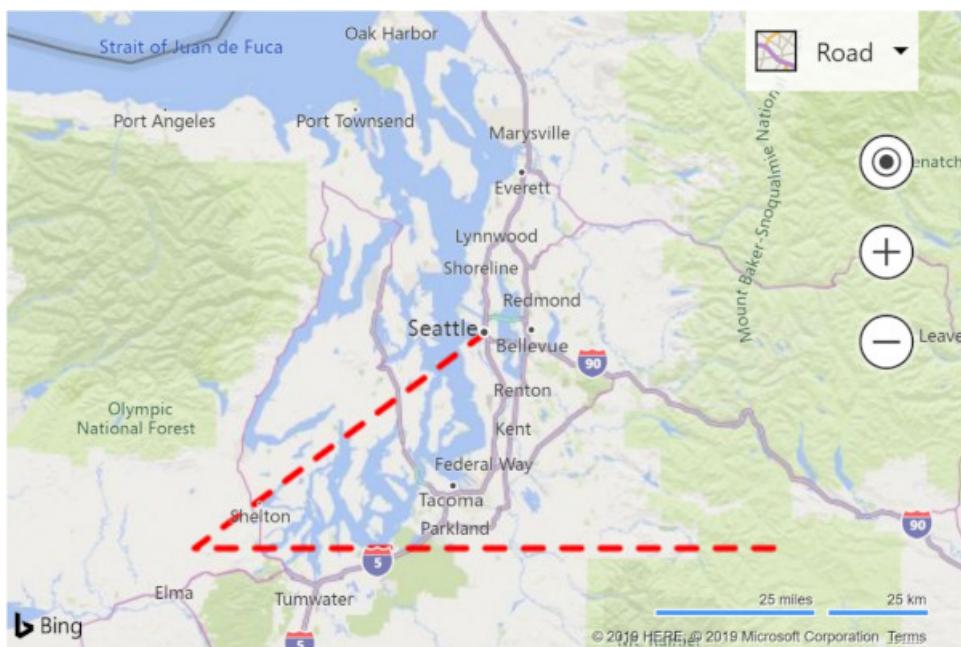
```

//Get the center of the map.
var center = map.getCenter();

//Create the polyline.
var polyline = new Microsoft.Maps.Polyline([
    center,
    new Microsoft.Maps.Location(center.latitude - 0.5, center.longitude - 1),
    new Microsoft.Maps.Location(center.latitude - 0.5, center.longitude + 1)
], {
    strokeColor: 'red',
    strokeThickness: 4,
    strokeDashArray: [3, 3]
});

//Add the polyline to the map using a layer.
var layer = new Microsoft.Maps.Layer();
layer.add(polyline);
map.layers.insert(layer);

```



After: Azure Maps

In Azure Maps, polylines are referred to the more commonly geospatial terms [LineString](#) or [MultiLineString](#) objects. These objects can be added to a data source and rendered using a line layer. The stroke color, width and dash array options are nearly identical between the platforms.

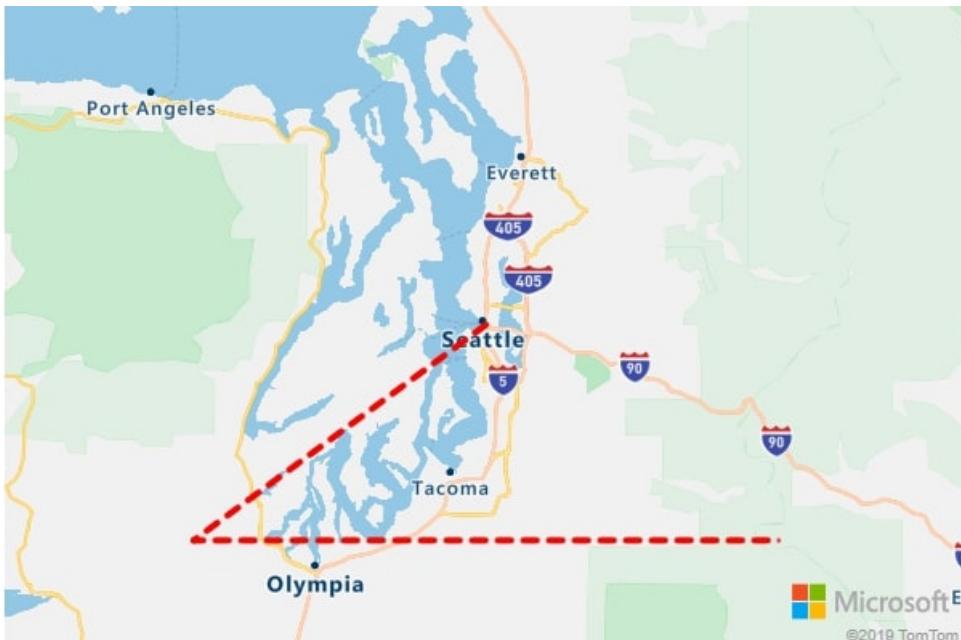
```

//Get the center of the map.
var center = map.getCamera().center;

//Create a data source and add it to the map.
var datasource = new atlas.source.DataSource();
map.sources.add(datasource);

//Create a line and add it to the data source.
datasource.add(new atlas.data.LineString([
    center,
    [center[0] - 1, center[1] - 0.5],
    [center[0] + 1, center[1] - 0.5]
]));

//Add a layer for rendering line data.
map.layers.add(new atlas.layer.LineLayer(datasource, null, {
    strokeColor: 'red',
    strokeWidth: 4,
    strokeDashArray: [3, 3]
}));
```



Additional resources

- [Add lines to the map](#)
- [Line layer options](#)
- [Use data-driven style expressions](#)

Adding a polygon

Polygons are used to represent an area on the map. Azure Maps and Bing Maps provide very similar support for polygons. The examples below show how to create a polygon that forms a triangle based on the center coordinate of the map.

Before: Bing Maps

In Bing Maps, the `Polygon` class takes in an array of coordinates or coordinate rings and a set of options.

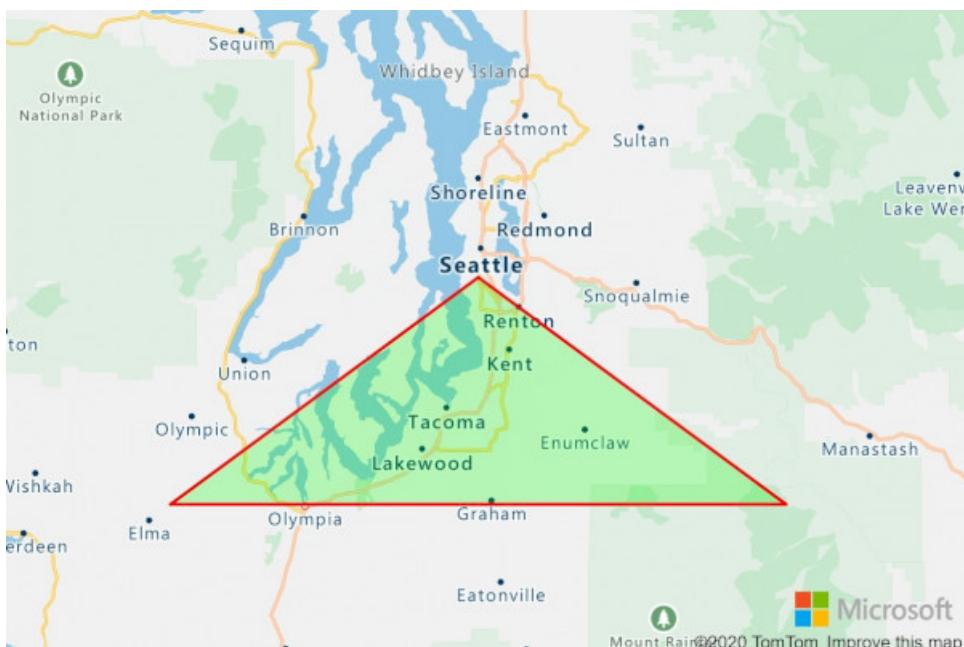
```

//Get the center of the map.
var center = map.getCenter();

//Create the polygon.
var polygon = new Microsoft.Maps.Polygon([
    center,
    new Microsoft.Maps.Location(center.latitude - 0.5, center.longitude - 1),
    new Microsoft.Maps.Location(center.latitude - 0.5, center.longitude + 1),
    center
], {
    fillColor: 'rgba(0, 255, 0, 0.5)',
    strokeColor: 'red',
    strokeThickness: 2
});

//Add the polygon to the map using a layer.
var layer = new Microsoft.Maps.Layer();
layer.add(polygon);
map.layers.insert(layer);

```



After: Azure Maps

In Azure Maps, Polygon and MultiPolygon objects can be added to a data source and rendered on the map using layers. The area of a polygon can be rendered in a polygon layer. The outline of a polygon can be rendered using a line layer.

```

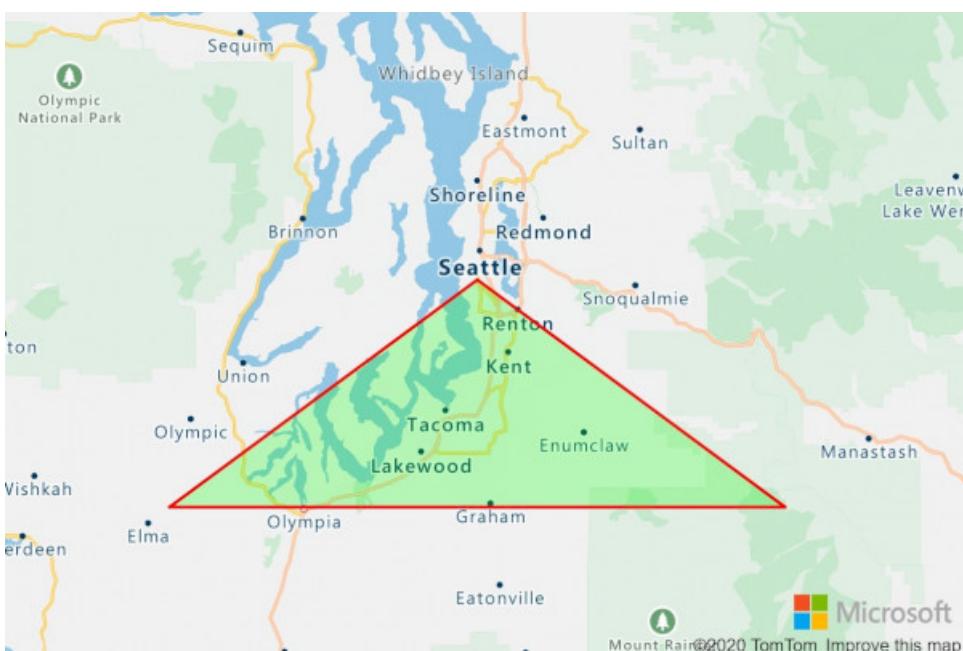
//Get the center of the map.
var center = map.getCamera().center;

//Create a data source and add it to the map.
datasource = new atlas.source.DataSource();
map.sources.add(datasource);

//Create a polygon and add it to the data source.
datasource.add(new atlas.data.Polygon([
    center,
    [center[0] - 1, center[1] - 0.5],
    [center[0] + 1, center[1] - 0.5],
    center
]));
//Add a polygon layer for rendering the polygon area.
map.layers.add(new atlas.layer.PolygonLayer(datasource, null, {
    fillColor: 'rgba(0, 255, 0, 0.5)'
}));

//Add a line layer for rendering the polygon outline.
map.layers.add(new atlas.layer.LineLayer(datasource, null, {
    strokeColor: 'red',
    strokeWidth: 2
}));

```



Additional resources

- [Add a polygon to the map](#)
- [Add a circle to the map](#)
- [Polygon layer options](#)
- [Line layer options](#)
- [Use data-driven style expressions](#)

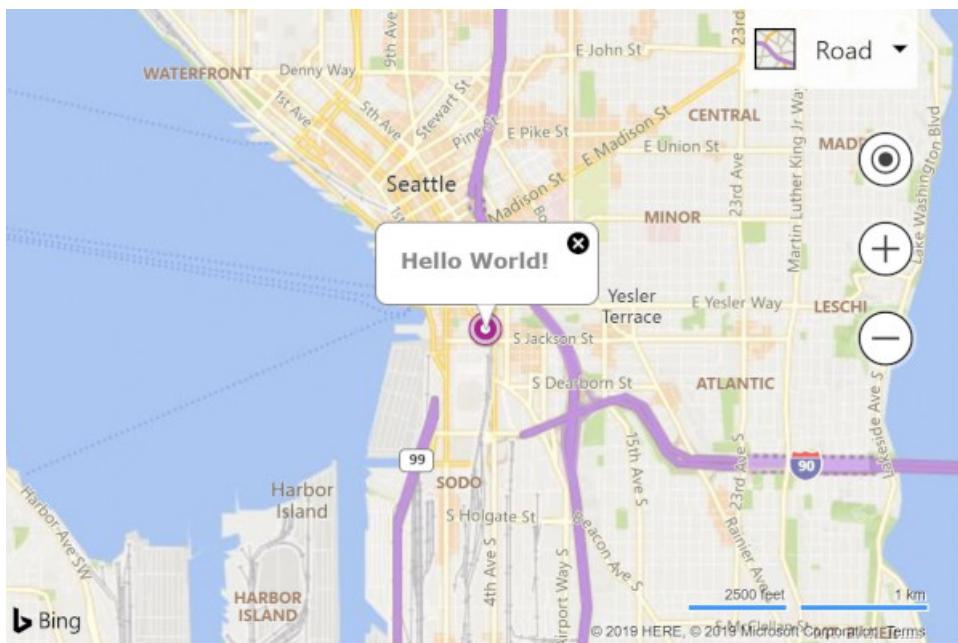
Display an infobox

Additional information for an entity can be displayed on the map as an `Microsoft.Maps.Infobox` class in Bing Maps, in Azure Maps this can be achieved using the `atlas.Popup` class. The examples below add a pushpin/marker to the map, and when clicked, displays an infobox/popup.

Before: Bing Maps

With Bing Maps, an infobox is created using the `Microsoft.Maps.Infobox` constructor.

```
//Add a pushpin where we want to display an infobox.  
var pushpin = new Microsoft.Maps.Pushpin(new Microsoft.Maps.Location(47.6, -122.33));  
  
//Add the pushpin to the map using a layer.  
var layer = new Microsoft.Maps.Layer();  
layer.add(pushpin);  
map.layers.insert(layer);  
  
//Create an infobox and bind it to the map.  
var infobox = new Microsoft.Maps.Infobox(new Microsoft.Maps.Location(47.6, -122.33), {  
    description: '<div style="padding:5px"><b>Hello World!</b></div>',  
    visible: false  
});  
infobox.setMap(map);  
  
//Add a click event to the pushpin to open the infobox.  
Microsoft.Maps.Events.addHandler(pushpin, 'click', function () {  
    infobox.setOptions({ visible: true });  
});
```



After: Azure Maps

In Azure Maps, a popup can be used to display additional information for a location. An `HTML` `string` or `HTMLElement` object can be passed into the `content` option of the popup. Popups can be displayed independently of any shape if desired and thus require a `position` value to be specified. To display a popup, call the `open` function and pass in the map that the popup is to be displayed on.

```

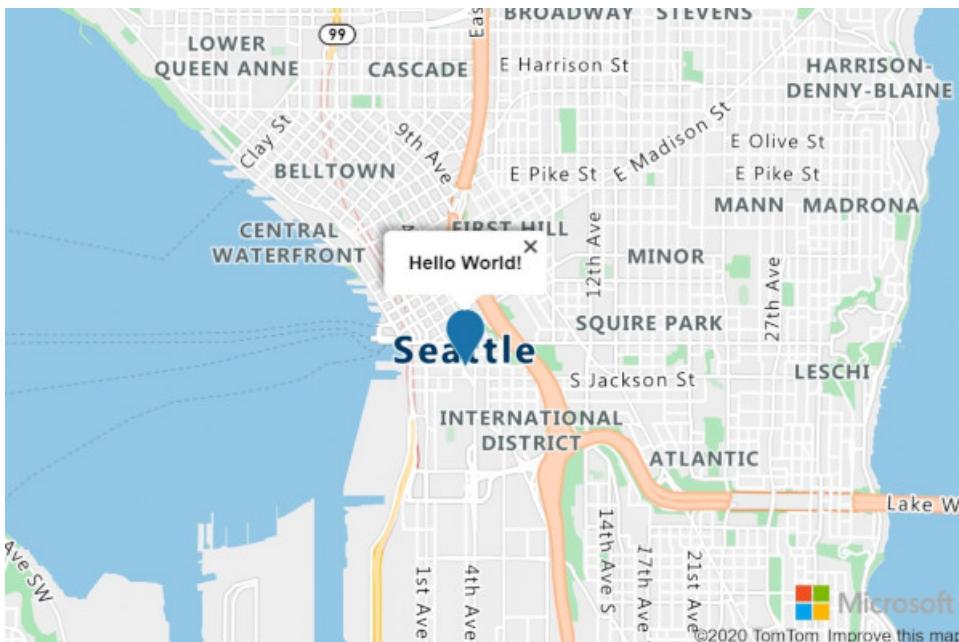
//Add a marker to the map that to display a popup for.
var marker = new atlas.HtmlMarker({
    position: [-122.33, 47.6]
});

//Add the marker to the map.
map.markers.add(marker);

//Create a popup.
var popup = new atlas.Popup({
    content: '<div style="padding:10px"><b>Hello World!</b></div>',
    position: [-122.33, 47.6],
    pixelOffset: [0, -35]
});

//Add a click event to the marker to open the popup.
map.events.add('click', marker, function () {
    //Open the popup
    popup.open(map);
});

```



NOTE

To do the same thing with a symbol, bubble, line or polygon layer, pass the layer into the maps event code instead of a marker.

Additional resources

- [Add a popup](#)
- [Popup with Media Content](#)
- [Popups on Shapes](#)
- [Reusing Popup with Multiple Pins](#)
- [Popup class](#)
- [Popup options](#)

Pushpin clustering

When visualizing many data points on the map, points overlap each other, the map looks cluttered and it becomes difficult to see and use. Clustering of point data can be used to improve this user experience and also

improve performance. Clustering point data is the process of combining point data that are near each other and representing them on the map as a single clustered data point. As the user zooms into the map, the clusters break apart into their individual data points.

The examples below load a GeoJSON feed of earthquake data from the past week and add it to the map. Clusters are rendered as scaled and colored circles depending on the number of points they contain.

NOTE

There are several different algorithms used for pushpin clustering. Bing Maps uses a simple grid-based function, while Azure Maps uses a more advanced and visually appealing point-based clustering methods.

Before: Bing Maps

In Bing Maps, GeoJSON data can be loaded using the GeoJSON module. Pushpins can be clustered by loading in the clustering module and using the clustering layer it contains.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <script type='text/javascript'>
        var map;
        var earthquakeFeed = 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.geojson';

        function initMap() {
            map = new Microsoft.Maps.Map(document.getElementById('myMap'), {
                credentials: '<Your Bing Maps Key>',
                center: new Microsoft.Maps.Location(20, -160),
                zoom: 2
            });

            //Load the GeoJSON and Clustering modules.
            Microsoft.Maps.loadModule(['Microsoft.Maps.GeoJson', 'Microsoft.Maps.Clustering'], function () {

                //Load the GeoJSON data from a URL.
                Microsoft.Maps.GeoJson.readFromUrl(earthquakeFeed, function (pins) {

                    //Create a ClusterLayer with options and add it to the map.
                    clusterLayer = new Microsoft.Maps.ClusterLayer(pins, {
                        clusteredPinCallback: createCustomClusteredPin,
                        gridSize: 100
                    });

                    map.layers.insert(clusterLayer);
                });
            });
        }

        //A function that defines how clustered pins are rendered.
        function createCustomClusteredPin(cluster) {
            //Get the number of pushpins in the cluster
            var clusterSize = cluster.containedPushpins.length;

            var radius = 20;      //Default radius to 20 pixels.
            var fillColor = 'lime'; //Default to lime green.

            if (clusterSize >= 750) {
                radius = 40;    //If point_count >= 750, radius is 40 pixels.
                fillColor = 'red'; //If the point_count >= 750, color is red.
            }
        }
    </script>

```

```

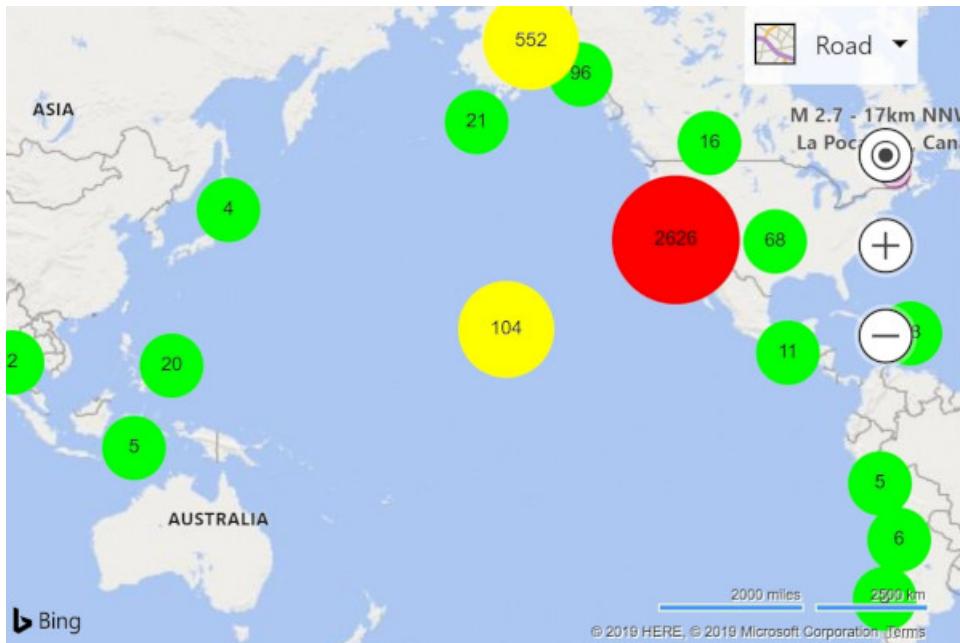
        } else if (clusterSize >= 100) {
            radius = 30;      //If point_count >= 100, radius is 30 pixels.
            fillColor = 'yellow';    //If the point_count >= 100, color is yellow.
        }

        //Create an SVG string of a circle with the specified radius and color.
        var svg = ['<svg xmlns="http://www.w3.org/2000/svg" width="", (radius * 2), "" height="",
(radius * 2), "">',
'<circle cx="", radius, "" cy="", radius, "" r="", radius, "" fill="", fillColor, ""/>',
'<text x="50%" y="50%" dominant-baseline="middle" text-anchor="middle" style="font-size:12px;font-family:arial;fill:black;" >{text}</text>',
'</svg>'];

        //Customize the clustered pushpin using the generated SVG and anchor on its center.
        cluster.setOptions({
            icon: svg.join(''),
            anchor: new Microsoft.Maps.Point(radius, radius),
            textOffset: new Microsoft.Maps.Point(0, radius - 8) //Subtract 8 to compensate for height of
text.
        });
    }
</script>

<!-- Bing Maps Script Reference -->
<script src="https://www.bing.com/api/maps/mapcontrol?callback=initMap" async defer></script>
</head>
<body>
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



After: Azure Maps

In Azure Maps, data is added and managed by a data source. Layers connect to data sources and render the data in them. The `DataSource` class in Azure Maps provides several clustering options.

- `cluster` – Tells the data source to cluster point data.
- `clusterRadius` - The radius in pixels to cluster points together.
- `clusterMaxZoom` - The maximum zoom level that clustering occurs. If you zoom in more than this, all points are rendered as symbols.
- `clusterProperties` - Defines custom properties that are calculated using expressions against all the points within each cluster and added to the properties of each cluster point.

When clustering is enabled, the data source will send clustered and unclustered data points to layers for rendering. The data source is capable of clustering hundreds of thousands of data points. A clustered data point has the following properties on it:

PROPERTY NAME	TYPE	DESCRIPTION
<code>cluster</code>	boolean	Indicates if feature represents a cluster.
<code>cluster_id</code>	string	A unique ID for the cluster that can be used with the <code>DataSource</code> classes <code>getClusterExpansionZoom</code> , <code>getClusterChildren</code> , and <code>getClusterLeaves</code> functions.
<code>point_count</code>	number	The number of points the cluster contains.
<code>point_count_abbreviated</code>	string	A string that abbreviates the <code>point_count</code> value if it is long. (for example, 4,000 becomes 4K)

The `DataSource` class has the following helper function for accessing additional information about a cluster using the `cluster_id`.

FUNCTION	RETURN TYPE	DESCRIPTION
<code>getClusterChildren(clusterId: number)</code>	<code>Promise<Feature<Geometry, any> Shape></code>	Retrieves the children of the given cluster on the next zoom level. These children may be a combination of shapes and sub-clusters. The sub-clusters will be features with properties matching cluster properties.
<code>getClusterExpansionZoom(clusterId: number)</code>	<code>Promise<number></code>	Calculates a zoom level that the cluster will start expanding or break apart.
<code>getClusterLeaves(clusterId: number, limit: number, offset: number)</code>	<code>Promise<Feature<Geometry, any> Shape></code>	Retrieves all points in a cluster. Set the <code>limit</code> to return a subset of the points and use the <code>offset</code> to page through the points.

When rendering clustered data on the map, it is often easiest to use two or more layers. The example below uses three layers, a bubble layer for drawing scaled colored circles based on the size of the clusters, a symbol layer to render the cluster size as text, and a second symbol layer for rendering the unclustered points. There are many other ways to render clustered data in Azure Maps highlighted in the [Cluster point data documentation](#).

GeoJSON data can be directly imported in Azure Maps using the `importDataFromUrl` function on the `DataSource` class.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <!-- Add references to the Azure Maps Map control JavaScript and CSS files -->
```

```

<!-- Add references to the Azure Maps map control JavaScript and CSS files. -->
<link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css"
type="text/css" />
<script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

<script type='text/javascript'>
    var map, datasource;
    var earthquakeFeed = 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.geojson';

    function initMap() {
        //Initialize a map instance.
        map = new atlas.Map('myMap', {
            center: [-160, 20],
            zoom: 1,
            authOptions: {
                authType: 'subscriptionKey',
                subscriptionKey: '<Your Azure Maps Key>'
            }
        });

        //Wait until the map resources are ready.
        map.events.add('ready', function () {

            //Create a data source and add it to the map.
            datasource = new atlas.source.DataSource(null, {
                //Tell the data source to cluster point data.
                cluster: true
            });
            map.sources.add(datasource);

            //Create layers for rendering clusters, their counts and unclustered points and add the
            layers to the map.
            map.layers.add([
                //Create a bubble layer for rendering clustered data points.
                new atlas.layer.BubbleLayer(datasource, null, {
                    //Scale the size of the clustered bubble based on the number of points inthe
                    cluster.
                    radius: [
                        'step',
                        ['get', 'point_count'],
                        20,           //Default of 20 pixel radius.
                        100, 30,     //If point_count >= 100, radius is 30 pixels.
                        750, 40      //If point_count >= 750, radius is 40 pixels.
                    ],
                    //Change the color of the cluster based on the value on the point_cluster property
                    color: [
                        'step',
                        ['get', 'point_count'],
                        'lime',          //Default to lime green.
                        100, 'yellow',   //If the point_count >= 100, color is yellow.
                        750, 'red'       //If the point_count >= 750, color is red.
                    ],
                    strokeWidth: 0,
                    filter: ['has', 'point_count'] //Only rendered data points that have a point_count
                    property, which clusters do.
                }),
                //Create a symbol layer to render the count of locations in a cluster.
                new atlas.layer.SymbolLayer(datasource, null, {
                    iconOptions: {
                        image: 'none' //Hide the icon image.
                    },
                    textOptions: {
                        textField: ['get', 'point_count_abbreviated'],
                        ...
                    }
                })
            ]);
        });
    }
</script>

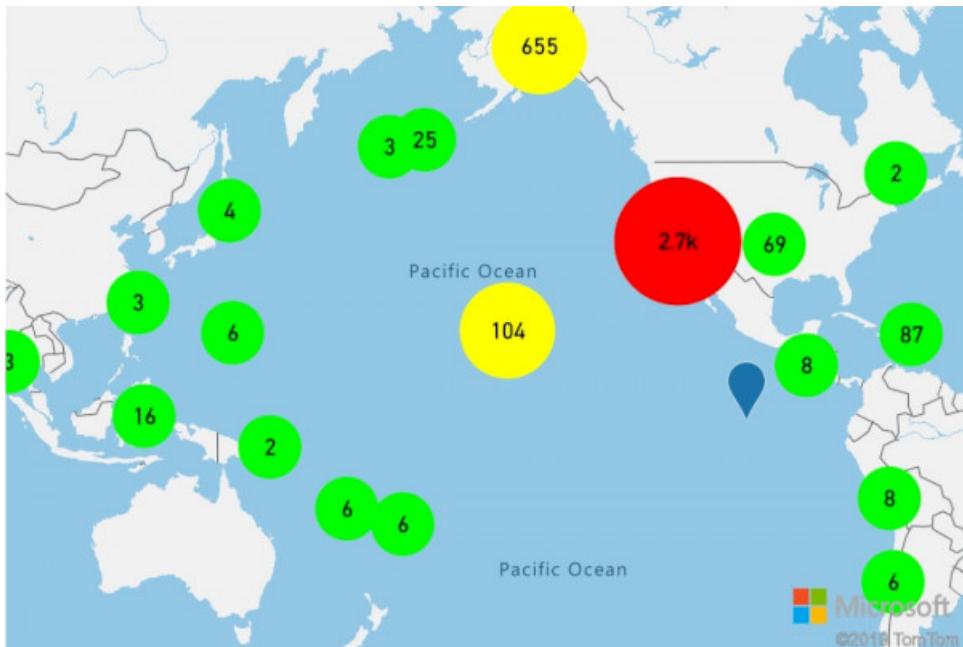
```

```

        offset: [0, 0.4]
    },
    //Create a layer to render the individual locations.
    new atlas.layer.SymbolLayer(datasource, null, {
        filter: ['!', ['has', 'point_count']] //Filter out clustered points from this layer.
    })
]);

//Retrieve a GeoJSON data set and add it to the data source.
datasource.importDataFromUrl(earthquakeFeed);
});
}
</script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



Additional resources

- [Add a Symbol layer](#)
- [Add a Bubble layer](#)
- [Cluster point data](#)
- [Use data-driven style expressions](#)

Add a heat map

Heat maps, also known as point density maps, are a type of data visualization used to represent the density of data using a range of colors. They're often used to show the data "hot spots" on a map and are a great way to render large point data sets.

The examples below load a GeoJSON feed of all earthquakes over the past month from the USGS and renders them as a heat map.

Before: Bing Maps

In Bing Maps, to create a heat map, load in the heat map module. Similarly, the GeoJSON module is loaded to add support for GeoJSON data.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <script type='text/javascript'>
        var map;
        var earthquakeFeed = 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_month.geojson';

        function initMap() {
            map = new Microsoft.Maps.Map(document.getElementById('myMap'), {
                credentials: '<Your Bing Maps Key>',
                center: new Microsoft.Maps.Location(20, -160),
                zoom: 2,
                mapTypeId: Microsoft.Maps.MapTypeId.aerial
            });

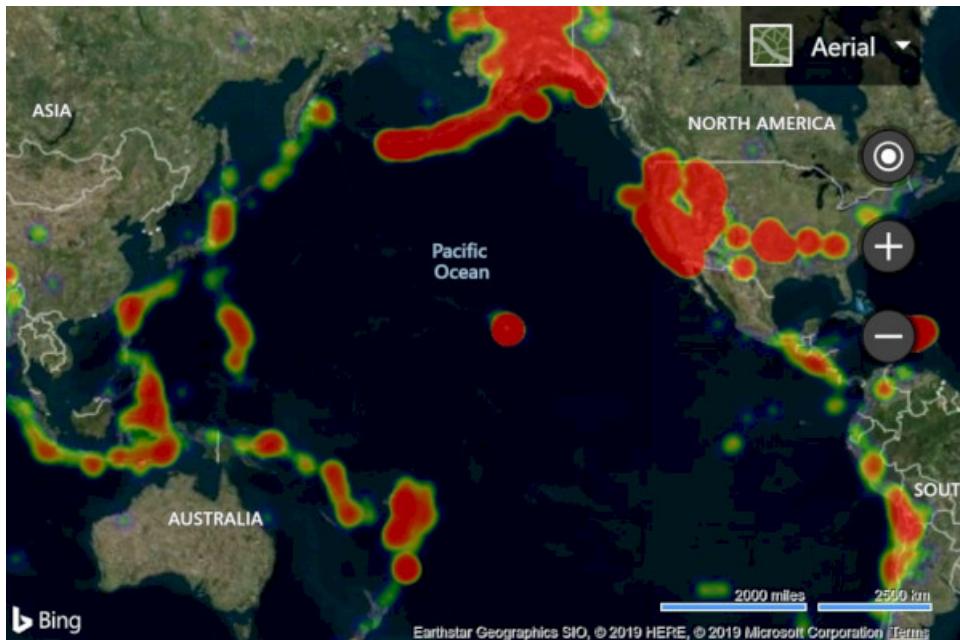
            //Load the GeoJSON and HeatMap modules.
            Microsoft.Maps.loadModule(['Microsoft.Maps.GeoJson', 'Microsoft.Maps.HeatMap'], function () {

                //Load the GeoJSON data from a URL.
                Microsoft.Maps.GeoJson.readFromUrl(earthquakeFeed, function (shapes) {

                    //Create a heat map and add it to the map.
                    var heatMap = new Microsoft.Maps.HeatMapLayer(shapes, {
                        opacity: 0.65,
                        radius: 10
                    });
                    map.layers.insert(heatMap);
                });
            });
        }
    </script>

    <!-- Bing Maps Script Reference -->
    <script src="https://www.bing.com/api/maps/mapcontrol?callback=initMap" async defer></script>
</head>
<body>
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



After: Azure Maps

In Azure Maps, load the GeoJSON data into a data source and connect the data source to a heat map layer. GeoJSON data can be directly imported in Azure Maps using the `importDataFromUrl` function on the `DataSource` class.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <script type='text/javascript'>
        var map;
        var earthquakeFeed = 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_month.geojson';

        function initMap() {
            //Initialize a map instance.
            map = new atlas.Map('myMap', {
                center: [-160, 20],
                zoom: 1,
                style: 'satellite_with_roads',

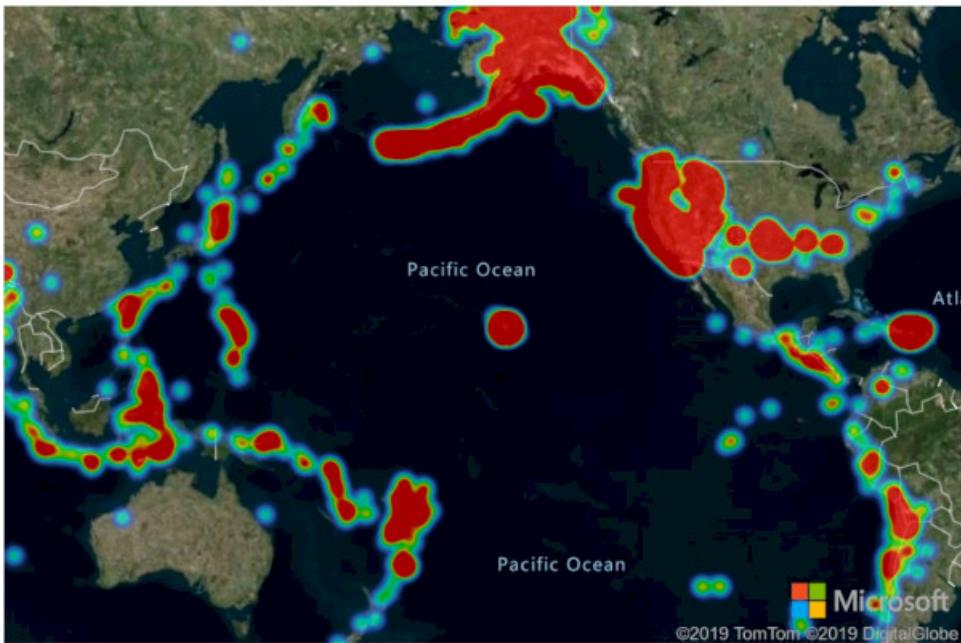
                //Add your Azure Maps key to the map SDK. Get an Azure Maps key at https://azure.com/maps.
                NOTE: The primary key should be used as the key.
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

            //Wait until the map resources are ready.
            map.events.add('ready', function () {

                //Create a data source and add it to the map.
                datasource = new atlas.source.DataSource();
                map.sources.add(datasource);

                //Load the earthquake data.
                datasource.importDataFromUrl(earthquakeFeed);

                //Create a layer to render the data points as a heat map.
                map.layers.add(new atlas.layer.HeatMapLayer(datasource, null, {
                    opacity: 0.65,
                    radius: 10
                }));
            });
        }
    </script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>
```



Additional resources

- [Add a heat map layer](#)
- [Heat map layer class](#)
- [Heat map layer options](#)
- [Use data-driven style expressions](#)

Overlay a tile layer

Tile layers allow you to overlay large images that have been broken up into smaller tiled images that align with the maps tiling system. This is a common way to overlay large images or very large data sets.

The examples below overlay a weather radar tile layer from Iowa Environmental Mesonet of Iowa State University that uses an X, Y, Zoom tiling naming schema.

Before: Bing Maps

In Bing Maps, tile layers can be created by using the `Microsoft.Maps.TileLayer` class.

```
var weatherTileLayer = new Microsoft.Maps.TileLayer({
    mercator: new Microsoft.Maps.TileSource({
        uriConstructor: 'https://mesonet.agron.iastate.edu/cache/tile.py/1.0.0/nexrad-n0q-
900913/{zoom}/{x}/{y}.png'
    })
});
map.layers.insert(weatherTileLayer);
```



After: Azure Maps

In Azure Maps, a tile layer can be added to the map in much the same way as any other layer. A formatted URL that has in x, y, zoom placeholders; `{x}`, `{y}`, `{z}` respectively is used to tell the layer where to access the tiles. Azure Maps tile layers also support `{quadkey}`, `{bbox-epsg-3857}` and `{subdomain}` placeholders.

TIP

In Azure Maps layers can easily be rendered below other layers, including base map layers. Often it is desirable to render tile layers below the map labels so that they are easy to read. The `map.layers.add` function takes in a second parameter that is the ID of a second layer to insert the new layer below. To insert a tile layer below the map labels the following code can be used:

```
map.layers.add(myTileLayer, "labels");
```

```
//Create a tile layer and add it to the map below the label layer.  
map.layers.add(new atlas.layer.TileLayer({  
    tileUrl: 'https://mesonet.agron.iastate.edu/cache/tile.py/1.0.0/nexrad-n0q-900913/{z}/{x}/{y}.png',  
    opacity: 0.8,  
    tileSize: 256  
}), 'labels');
```



TIP

Tile requests can be captured using the `transformRequest` option of the map. This will allow you to modify or add headers to the request if desired.

Additional resources

- [Add tile layers](#)
- [Tile layer class](#)
- [Tile layer options](#)

Show traffic data

Traffic data can be overlaid both Bing and Azure maps.

Before: Bing Maps

In Bing Maps, traffic data can be overlaid the map using the traffic module.

```
Microsoft.Maps.loadModule('Microsoft.Maps.Traffic', function () {
    var manager = new Microsoft.Maps.Traffic.TrafficManager(map);
    manager.show();
});
```



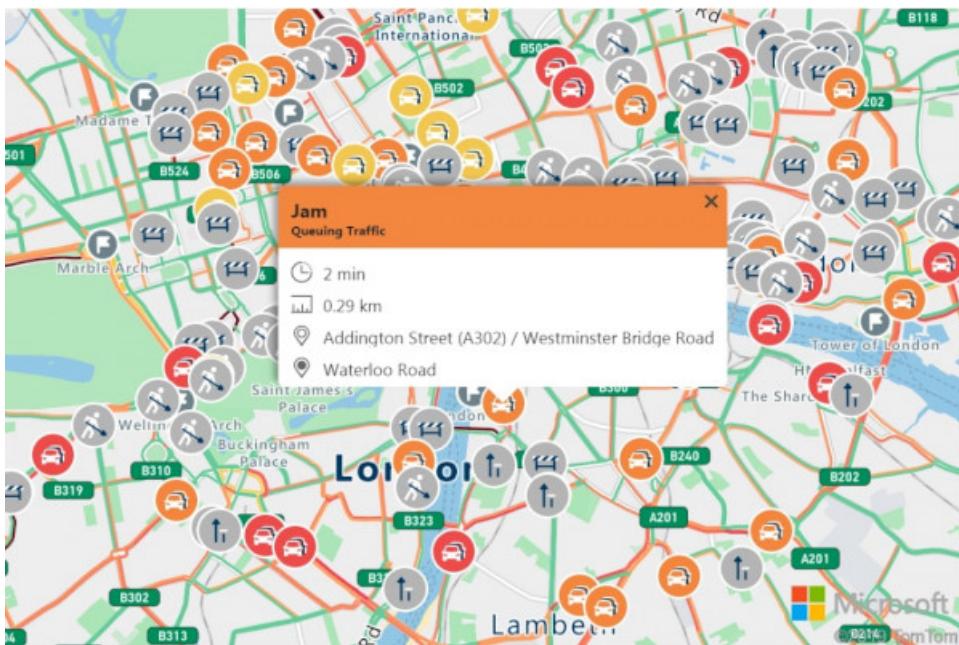
After: Azure Maps

Azure Maps provides several different options for displaying traffic. Traffic incidents, such as road closures and accidents can be displayed as icons on the map. Traffic flow, color coded roads, can be overlaid on the map and the colors can be modified to be based relative to the posted speed limit, relative to the normal expected delay, or absolute delay. Incident data in Azure Maps is updated every minute and flow data every 2 minutes.

```
map.setTraffic({
    incidents: true,
    flow: 'relative'
});
```



If you click on one of the traffic icons in Azure Maps, additional information is displayed in a popup.



Additional resources

- [Show traffic on the map](#)
- [Traffic overlay options](#)
- [Traffic control](#)

Add a ground overlay

Both Bing and Azure maps support overlaying georeferenced images on the map so that they move and scale as you pan and zoom the map. In Bing Maps these are known as ground overlays while in Azure Maps they are referred to as image layers. These are great for building floor plans, overlaying old maps, or imagery from a drone.

Before: Bing Maps

When creating a ground overlay in Bing Maps you need to specify the URL to the image to overlay and a bounding box to bind the image to on the map. This example overlays a map image of [Newark New Jersey from 1922](#) on the map.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <script type='text/javascript'>
        var map;

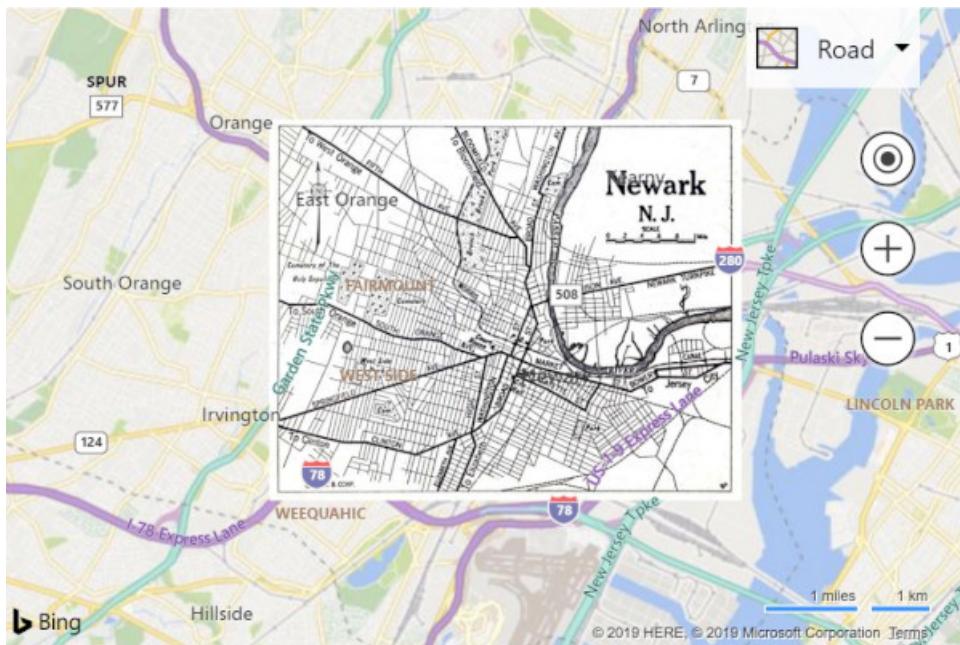
        function initMap() {
            map = new Microsoft.Maps.Map(document.getElementById('myMap'), {
                credentials: '<Your Bing Maps Key>',
                center: new Microsoft.Maps.Location(40.740, -74.18),
                zoom: 12
            });

            var overlay = new Microsoft.Maps.GroundOverlay({
                //Create a LocationRect from the edges of the bounding box; north, west, south, east.
                bounds: Microsoft.Maps.LocationRect.fromEdges(40.773941, -74.22655, 40.712216, -74.12544),
                imageUrl: 'newark_nj_1922.jpg'
            });
            map.layers.insert(overlay);
        }
    </script>

    <!-- Bing Maps Script Reference -->
    <script src="https://www.bing.com/api/maps/mapcontrol?callback=initMap" async defer></script>
</head>
<body>
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```

Running this code in a browser will display a map that looks like the following image:



After: Azure Maps

In Azure Maps, georeferenced images can be overlaid using the `atlas.layer.ImageLayer` class. This class requires a URL to an image and a set of coordinates for the four corners of the image. The image must be hosted either on the same domain or have CORs enabled.

TIP

If you only have north, south, east, west and rotation information, and not coordinates for each corner of the image, you can use the static `atlas.layer.ImageLayer.getCoordinatesFromEdges` function.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css"
type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <script type='text/javascript'>
        var map;

        function initMap() {
            //Initialize a map instance.
            map = new atlas.Map('myMap', {
                center: [-74.18, 40.740],
                zoom: 12,
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

            //Wait until the map resources are ready.
            map.events.add('ready', function () {

                //Create an image layer and add it to the map.
                map.layers.add(new atlas.layer.ImageLayer({
                    url: 'newark_nj_1922.jpg',
                    coordinates: [
                        [-74.22655, 40.773941], //Top Left Corner
                        [-74.12544, 40.773941], //Top Right Corner
                        [-74.12544, 40.712216], //Bottom Right Corner
                        [-74.22655, 40.712216] //Bottom Left Corner
                    ]
                }));
            });
        }
    </script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>
```



Additional resources

- [Overlay an image](#)
- [Image layer class](#)

Add KML data to the map

Both Azure and Bing maps can import and render KML, KMZ, GeoRSS, GeoJSON and Well-Known Text (WKT) data on the map. Azure Maps also supports GPX, GML, spatial CSV files, Web-Mapping Services (WMS), Web-Mapping Tile Services (WMPS), and Web Feature Services (WFS).

Before: Bing Maps

Running this code in a browser will display a map that looks like the following image:

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

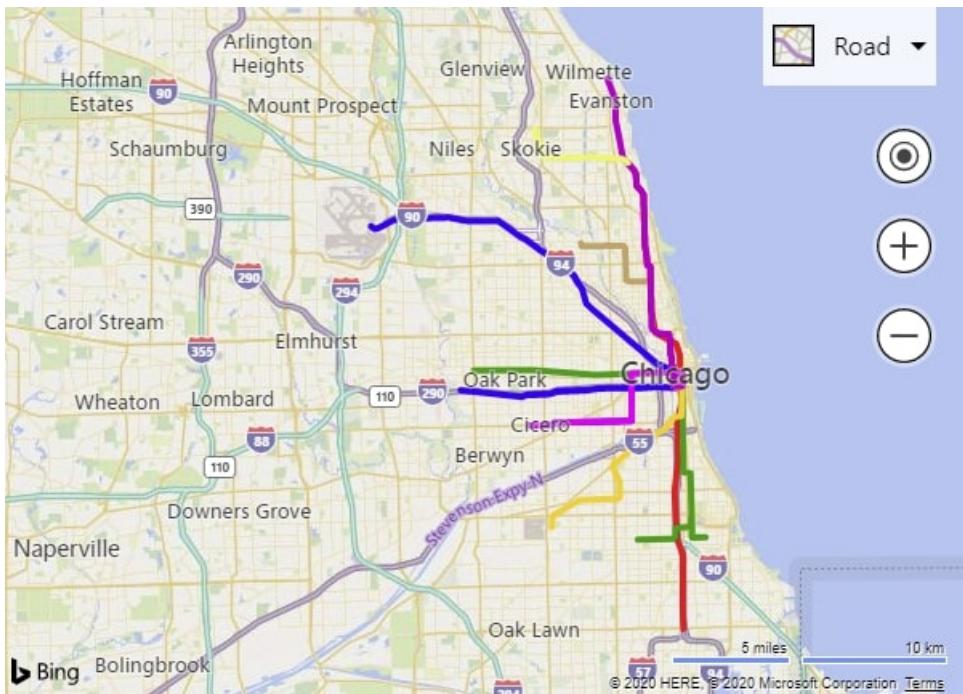
    <script type='text/javascript'>
        var map;

        function initMap() {
            map = new Microsoft.Maps.Map('#myMap', {
                credentials: '<Your Bing Maps Key>',
                center: new Microsoft.Maps.Location(40.747, -73.985),
                zoom: 12
            });

            Microsoft.Maps.loadModule('Microsoft.Maps.GeoXml', function () {
                var callback = function (dataset) {
                    if (dataset.shapes) {
                        var l = new Microsoft.Maps.Layer();
                        l.add(dataset.shapes);
                        map.layers.insert(l);
                    }
                    if (dataset.layers) {
                        for (var i = 0, len = dataset.layers.length; i < len; i++) {
                            map.layers.insert(dataset.layers[i]);
                        }
                    }
                };
                Microsoft.Maps.GeoXml.readFromUrl('myKMLFile.kml', { error: function (msg) { alert(msg); } }, callback);
            });
        }
    </script>

    <!-- Bing Maps Script Reference -->
    <script src="https://www.bing.com/api/maps/mapcontrol?callback=initMap" async defer></script>
</head>
<body>
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



After: Azure Maps

In Azure Maps, GeoJSON is the main data format used in the web SDK, additional spatial data formats can be easily integrated in using the [spatial IO module](#). This module has functions for both reading and writing spatial data and also includes a simple data layer that can easily render data from any of these spatial data formats. To read the data in a spatial data file, pass in a URL, or raw data as string or blob into the `atlas.io.read` function. This will return all the parsed data from the file that can then be added to the map. KML is a bit more complex than most spatial data format as it includes a lot more styling information. The `SpatialDataLayer` class supports rendering majority of these styles, however icons images have to be loaded into the map before loading the feature data, and ground overlays have to be added as layers to the map separately. When loading data via a URL, it should be hosted on a CORs enabled endpoint, or a proxy service should be passed in as an option into the read function.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css"
        type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <!-- Add reference to the Azure Maps Spatial IO module. -->
    <script src="https://atlas.microsoft.com/sdk/javascript/spatial/0/atlas-spatial.js"></script>

    <script type='text/javascript'>
        var map, datasource, layer;

        function initMap() {
            //Initialize a map instance.
            map = new atlas.Map('myMap', {
                view: 'Auto',
                authOptions: {
                    authType: 'subscriptionKey',

```

```

        subscriptionKey: '<Your Azure Maps Key>'
    }
});

//Wait until the map resources are ready.
map.events.add('ready', function () {

    //Create a data source and add it to the map.
    datasource = new atlas.source.DataSource();
    map.sources.add(datasource);

    //Add a simple data layer for rendering the data.
    layer = new atlas.layer.SimpleDataLayer(datasource);
    map.layers.add(layer);

    //Read a KML file from a URL or pass in a raw KML string.
    atlas.io.read('myKMLFile.kml').then(async r => {
        if (r) {

            //Check to see if there are any icons in the data set that need to be loaded into
            the map resources.
            if (r.icons) {
                //For each icon image, create a promise to add it to the map, then run the
                promises in parallel.
                var imagePromises = [];

                //The keys are the names of each icon image.
                var keys = Object.keys(r.icons);

                if (keys.length !== 0) {
                    keys.forEach(function (key) {
                        imagePromises.push(map.imageSprite.add(key, r.icons[key]));
                    });

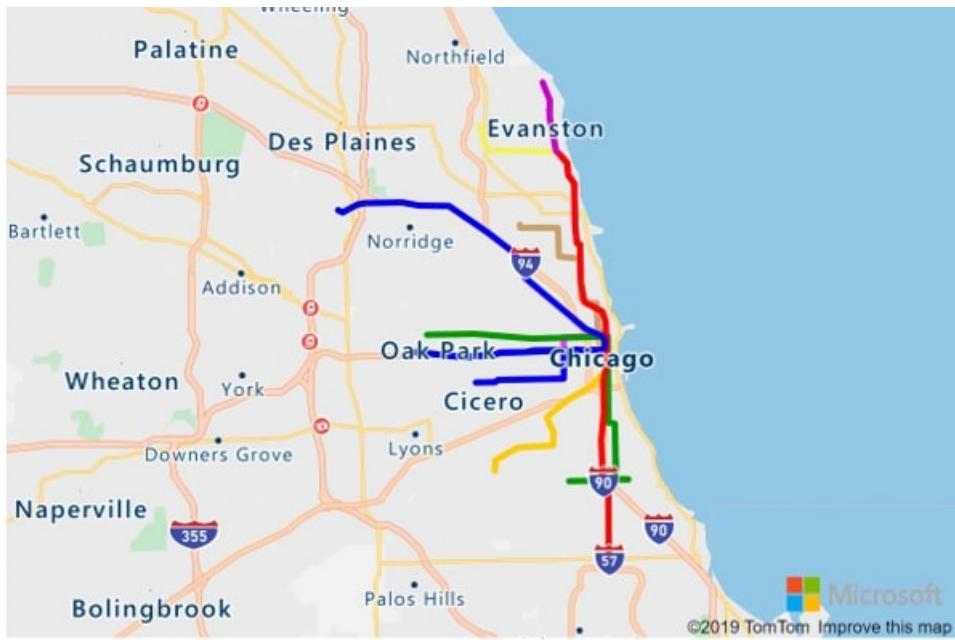
                    await Promise.all(imagePromises);
                }
            }

            //Load all features.
            if (r.features && r.features.length > 0) {
                datasource.add(r.features);
            }

            //Load all ground overlays.
            if (r.groundOverlays && r.groundOverlays.length > 0) {
                map.layers.add(r.groundOverlays);
            }

            //If bounding box information is known for data, set the map view to it.
            if (r.bbox) {
                map.setCamera({ bounds: r.bbox, padding: 50 });
            }
        }
    });
});

//
```



Additional resources

- [atlas.io.read function](#)
- [SimpleDataLayer](#)
- [SimpleDataLayerOptions](#)

Add drawing tools

Both Bing and Azure Maps provide a module that add the ability for the user to draw and edit shapes on the map using the mouse or other input device. They both support drawing pushpins, lines, and polygons. Azure Maps also provides options for drawing circles and rectangles.

Before: Bing Maps

In Bing Maps the `DrawingTools` module is loaded using the `Microsoft.Maps.loadModule` function. Once loaded, an instance of the `DrawingTools` class can be created and the `showDrawingManager` function is called add a toolbar to the map.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <script type='text/javascript'>
        var map, drawingManager;

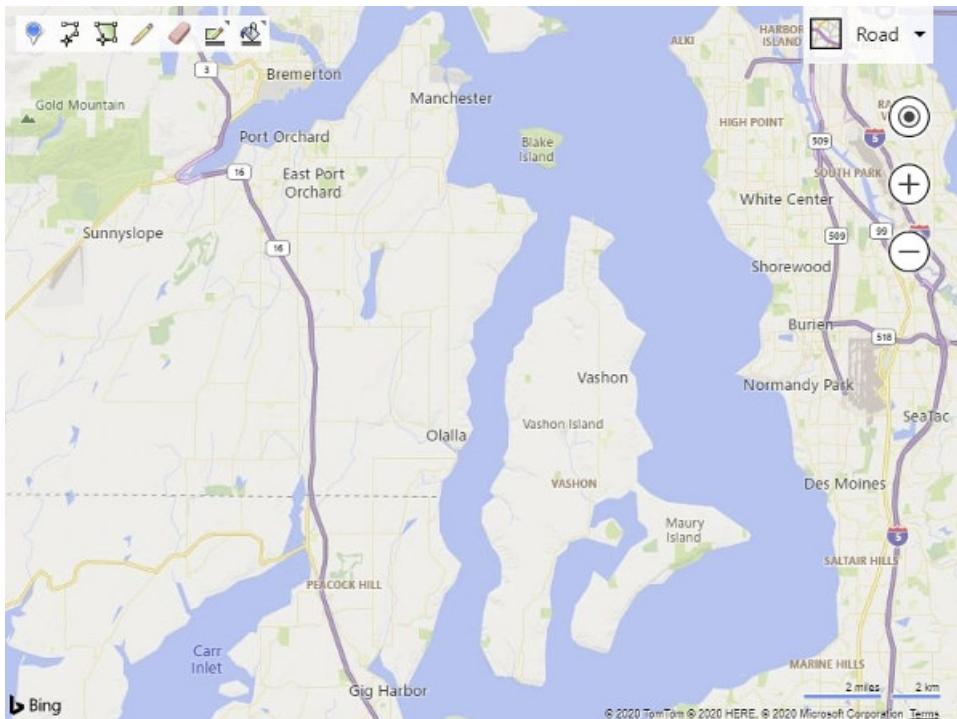
        function initMap() {
            map = new Microsoft.Maps.Map('#myMap', {
                credentials: '<Your Bing Maps Key>'
            });

            //Load the DrawingTools module
            Microsoft.Maps.loadModule('Microsoft.Maps.DrawingTools', function () {
                //Create an instance of the DrawingTools class and bind it to the map.
                var tools = new Microsoft.Maps.DrawingTools(map);

                //Show the drawing toolbar and enable edditing on the map.
                tools.showDrawingManager(function (manager) {
                    //Store a reference to the drawing manager as it will be useful later.
                    drawingManager = manager;
                });
            });
        }
    </script>

    <!-- Bing Maps Script Reference -->
    <script src="https://www.bing.com/api/maps/mapcontrol?callback=initMap" async defer></script>
</head>
<body>
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



After: Azure Maps

In Azure Maps the drawing tools module needs to be loaded by loading the JavaScript and CSS files need to be

referenced in the app. Once the map has loaded, an instance of the `DrawingManager` class can be created and a `DrawingToolbar` instance attached.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css"
type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <!-- Add references to the Azure Maps Map Drawing Tools JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/drawing/0/atlas-drawing.min.css"
type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/drawing/0/atlas-drawing.min.js"></script>

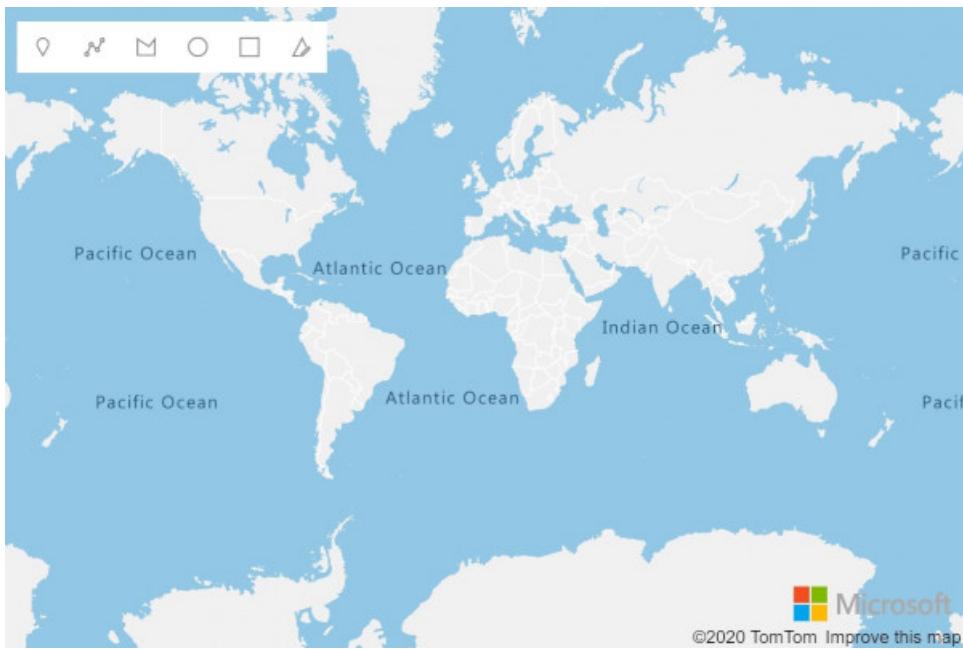
    <script type='text/javascript'>
        var map, drawingManager;

        function initMap() {
            //Initialize a map instance.
            map = new atlas.Map('myMap', {
                view: 'Auto',

                //Add your Azure Maps key to the map SDK. Get an Azure Maps key at https://azure.com/maps.
                NOTE: The primary key should be used as the key.
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

            //Wait until the map resources are ready.
            map.events.add('ready', function () {

                //Create an instance of the drawing manager and display the drawing toolbar.
                drawingManager = new atlas.drawing.DrawingManager(map, {
                    toolbar: new atlas.control.DrawingToolbar({ position: 'top-left' })
                });
            });
        }
    </script>
</head>
<body onload="initMap()">
    <div id="myMap" style="position:relative;width:600px;height:400px;"></div>
</body>
</html>
```



TIP

In Azure Maps layers the drawing tools provide multiple ways that users can draw shapes. For example, when drawing a polygon the user can click to add each point, or hold the left mouse button down and drag the mouse to draw a path. This can be modified using the `interactionType` option of the `DrawingManager`.

Additional resources

- [Documentation](#)
- [Code samples](#)

Additional resources

Take a look at the [open-source Azure Maps Web SDK modules](#). These modules provide a ton of additional functionality and are fully customizable.

Review code samples related migrating other Bing Maps features:

Data visualizations

[Contour layer](#)

[Data Binning](#)

Services

[Using the Azure Maps services module](#)

[Search for points of interest](#)

[Get information from a coordinate \(reverse geocode\)](#)

[Show directions from A to B](#)

[Search Autosuggest with JQuery UI](#)

Learn more about the Azure Maps Web SDK.

[How to use the map control](#)

[How to use the services module](#)

[How to use the drawing tools module](#)

[Code samples](#)

[Azure Maps Web SDK Service API reference documentation](#)

Clean up resources

No resources to be cleaned up.

Next steps

Learn more about migrating from Bing Maps to Azure Maps.

[Migrate a web service](#)

Tutorial: Migrate web service from Bing Maps

6/1/2021 • 34 minutes to read • [Edit Online](#)

Both Azure and Bing Maps provide access to spatial APIs through REST web services. The API interfaces for these platforms perform similar functionalities but use different naming conventions and response objects. In this tutorial, you will learn how to:

- Forward and reverse geocoding
- Search for points of interest
- Calculate routes and directions
- Retrieve a map image
- Calculate a distance matrix
- Get time zone details

The following table provides the Azure Maps service APIs that provide similar functionality to the listed Bing Maps service APIs.

BING MAPS SERVICE API	AZURE MAPS SERVICE API
Autosuggest	Search
Directions (including truck)	Route directions
Distance Matrix	Route Matrix
Imagery – Static Map	Render
Isochrones	Route Range
Local Insights	Search + Route Range
Local Search	Search
Location Recognition (POIs)	Search
Locations (forward/reverse geocoding)	Search
Snap to Road	POST Route directions
Spatial Data Services (SDS)	Search + Route + other Azure Services
Time Zone	Time Zone
Traffic Incidents	Traffic Incident Details
Elevation	Elevation

The following service APIs are not currently available in Azure Maps:

- Optimized Itinerary Routes - Planned. Azure Maps Route API does support traveling salesmen optimization for a single vehicle.
- Imagery Metadata – Primarily used for getting tile URLs in Bing Maps. Azure Maps has a standalone service for directly accessing map tiles.

Azure Maps has several additional REST web services that may be of interest;

- [Azure Maps Creator](#) – Create a custom private digital twin of buildings and spaces.
- [Spatial operations](#) – Offload complex spatial calculations and operations, such as geofencing, to a service.
- [Map Tiles](#) – Access road and imagery tiles from Azure Maps as raster and vector tiles.
- [Batch routing](#) – Allows up to 1,000 route requests to be made in a single batch over a period of time. Routes are calculated in parallel on the server for faster processing.
- [Traffic Flow](#) – Access real-time traffic flow data as both raster and vector tiles.
- [Geolocation API \(Preview\)](#) – Get the location of an IP address.
- [Weather Services](#) – Gain access to real-time and forecast weather data.

Be sure to also review the following best practices guides:

- [Best practices for search](#)
- [Best practices for routing](#)

Prerequisites

1. Sign in to the [Azure portal](#). If you don't have an Azure subscription, create a [free account](#) before you begin.
2. [Make an Azure Maps account](#)
3. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

Geocoding addresses

Geocoding is the process of converting an address (like "1 Microsoft way, Redmond, WA") into a coordinate (like longitude: -122.1298, latitude: 47.64005). Coordinates are then often used to position a pushpin on a map or center a map.

Azure Maps provides several methods for geocoding addresses;

- [Free-form address geocoding](#): Specify a single address string (like "1 Microsoft way, Redmond, WA") and process the request immediately. This service is recommended if you need to geocode individual addresses quickly.
- [Structured address geocoding](#): Specify the parts of a single address, such as the street name, city, country, and postal code and process the request immediately. This service is recommended if you need to geocode individual addresses quickly and the data is already parsed into its individual address parts.
- [Batch address geocoding](#): Create a request containing up to 10,000 addresses and have them processed over a period of time. All the addresses will be geocoded in parallel on the server and when completed the full result set can be downloaded. This service is recommended for geocoding large data sets.
- [Fuzzy search](#): This API combines address geocoding with point of interest search. This API takes in a free-form string that can be an address, place, landmark, point of interest, or point of interest category and process the request immediately. This API is recommended for applications where users can search for addresses or points of interest from the same textbox.
- [Fuzzy batch search](#): Create a request containing up to 10,000 addresses, places, landmarks, or point of interests and have them processed over a period of time. All the data will be processed in parallel on the server and when completed the full result set can be downloaded.

The following tables cross-reference the Bing Maps API parameters with the comparable API parameters in Azure Maps for structured and free-form address geocoding.

Location by Address (structured address)

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>addressLine</code>	<code>streetNumber</code> , <code>streetName</code> or <code>crossStreet</code>
<code>adminDistrict</code>	<code>countrySubdivision</code>
<code>countryRegion</code>	<code>country</code> and <code>countryCode</code>
<code>locality</code>	<code>municipality</code> or <code>municipalitySubdivision</code>
<code>postalCode</code>	<code>postalCode</code>
<code>maxResults</code> (<code>maxRes</code>)	<code>limit</code>
<code>includeNeighborhood</code> (<code>inlnb</code>)	N/A – Always returned by Azure Maps if available.
<code>include</code> (<code>incl</code>)	N/A – Country ISO2 Code always returned by Azure Maps.
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation.
<code>culture</code> (<code>c</code>)	<code>language</code> – See supported languages documentation.
<code>userRegion</code> (<code>ur</code>)	<code>view</code> – See supported views documentation.

Azure Maps also supports:

- `countrySecondarySubdivision` – County, districts
- `countryTertiarySubdivision` - Named areas; boroughs, cantons, communes
- `ofs` - Page through the results in combination with `maxResults` parameter.

Location by Query (free-form address string)

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>query</code>	<code>query</code>
<code>maxResults</code> (<code>maxRes</code>)	<code>limit</code>
<code>includeNeighborhood</code> (<code>inlnb</code>)	N/A – Always returned by Azure Maps if available.
<code>include</code> (<code>incl</code>)	N/A – Country ISO2 Code always returned by Azure Maps.
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation.
<code>culture</code> (<code>c</code>)	<code>language</code> – See supported languages documentation.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>userRegion</code> (<code>ur</code>)	<code>view</code> – See supported views documentation .

Azure Maps also supports;

- `typeahead` – Species if the query will be interpreted as a partial input and the search will enter predictive mode (autosuggest/autocomplete).
- `countrySet` – A comma-separated list of ISO2 countries codes in which to limit the search to.
- `lat / lon`, `topLeft / btmRight`, `radius` – Specify user location and area to make the results more locally relevant.
- `ofs` - Page through the results in combination with `maxResults` parameter.

An example of how to use the search service is documented [here](#). Be sure to review the [best practices for search documentation](#).

Reverse geocode a coordinate (Find a Location by Point)

Reverse geocoding is the process of converting geographic coordinates (like longitude: -122.1298, latitude: 47.64005) into its approximate address (like "1 Microsoft way, Redmond, WA").

Azure Maps provides several reverse geocoding methods;

- [Address reverse geocoder](#): Specify a single geographic coordinate to get its approximate address and process the request immediately.
- [Cross street reverse geocoder](#): Specify a single geographic coordinate to get nearby cross street information (for example, 1st & main) and process the request immediately.
- [Batch address reverse geocoder](#): Create a request containing up to 10,000 coordinates and have them processed over a period of time. All the data will be processed in parallel on the server and when completed the full result set can be downloaded.

The following table cross-references the Bing Maps API parameters with the comparable API parameters in Azure Maps.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>point</code>	<code>query</code>
<code>includeEntityTypes</code>	<code>entityType</code> – See entity type comparison table below.
<code>includeNeighborhood</code> (<code>inclnb</code>)	N/A – Always returned by Azure Maps if available.
<code>include</code> (<code>inlc</code>)	N/A – Country ISO2 Code always returned by Azure Maps.
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation .
<code>culture</code> (<code>c</code>)	<code>language</code> – See supported languages documentation .
<code>userRegion</code> (<code>ur</code>)	<code>view</code> – See supported views documentation .

Be sure to review the [best practices for search documentation](#).

The Azure Maps reverse geocoding API has some additional features not available in Bing Maps that might be

useful to integrate when migrating your app:

- Retrieve speed limit data.
- Retrieve road use information; local road, arterial, limited access, ramp, etc.
- The side of street the coordinate falls on.

Entity type comparison table

The following table cross references the Bing Maps entity type values to the equivalent property names in Azure Maps.

BING MAPS ENTITY TYPE	COMPARABLE AZURE MAPS ENTITY TYPE	DESCRIPTION
Address		<i>Address</i>
Neighborhood	Neighbourhood	<i>Neighborhood</i>
PopulatedPlace	Municipality or MunicipalitySubdivision	<i>City, Town or Sub, or Super City</i>
Postcode1	PostalCodeArea	<i>Postal Code or Zip Code</i>
AdminDivision1	CountrySubdivision	<i>State or Province</i>
AdminDivision2	CountrySecondarySubdivision	<i>County or districts</i>
CountryRegion	Country	<i>Country name</i>
	CountryTertiarySubdivision	<i>Boroughs, Cantons, Communes</i>

Get location suggestions (Autosuggest)

Several of the Azure Maps search API's support predictive mode that can be used for autosuggest scenarios. The Azure Maps [fuzzy search](#) API is the most like the Bing Maps Autosuggest API. The following API's also support predictive mode, add `&typeahead=true` to the query;

- [Free-form address geocoding](#): Specify a single address string (like `"1 Microsoft way, Redmond, WA"`) and process the request immediately. This service is recommended if you need to geocode individual addresses quickly.
- [Fuzzy search](#): This API combines address geocoding with point of interest search. This API takes in a free-form string that can be an address, place, landmark, point of interest, or point of interest category and process the request immediately. This API is recommended for applications where users can search for addresses or points of interest from the same textbox.
- [POI search](#): Search for points of interests by name. For example; `"starbucks"` .
- [POI category search](#): Search for points of interests by category. For example; "restaurant".

Calculate routes and directions

Azure Maps can be used to calculate routes and directions. Azure Maps has many of the same functionalities as the Bing Maps routing service, such as;

- arrival and departure times

- real-time and predictive based traffic routes
- different modes of transportation; driving, walking, truck
- waypoint order optimization (traveling salesmen)

NOTE

Azure Maps requires all waypoints to be coordinates. Addresses will need to be geocoded first.

The Azure Maps routing service provides the following APIs for calculating routes;

- [Calculate route](#): Calculate a route and have the request processed immediately. This API supports both GET and POST requests. POST requests are recommended when specifying a large number of waypoints or when using lots of the route options to ensure that the URL request doesn't become too long and cause issues.
- [Batch route](#): Create a request containing up to 1,000 route request and have them processed over a period of time. All the data will be processed in parallel on the server and when completed the full result set can be downloaded.
- [Mobility services \(Preview\)](#) : Calculate routes and directions using public transit.

The following table cross-references the Bing Maps API parameters with the comparable API parameters in Azure Maps.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>avoid</code>	<code>avoid</code>
<code>dateTime (dt)</code>	<code>departAt</code> or <code>arriveAt</code>
<code>distanceBeforeFirstTurn (dbft)</code>	N/A
<code>distanceUnit (du)</code>	N/A – Azure Maps only uses the metric system.
<code>heading (hd)</code>	<code>vehicleHeading</code>
<code>maxSolutions (maxSolns)</code>	<code>maxAlternatives</code> , <code>alternativeType</code> , <code>minDeviationDistance</code> , and <code>minDeviationTime</code>
<code>optimize (optwz)</code>	<code>routeType</code> and <code>traffic</code>
<code>optimizeWaypoints (optWp)</code>	<code>computeBestOrder</code>
<code>routeAttributes (ra)</code>	<code>instructionsType</code>
<code>routePathOutput (rpo)</code>	<code>routeRepresentation</code>
<code>timeType (tt)</code>	<code>departAt</code> or <code>arriveAt</code>
<code>tolerances (tl)</code>	N/A
<code>travelMode</code>	<code>travelMode</code>

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>waypoint.n</code> (<code>wp.n</code>) or <code>viaWaypoint.n</code> (<code>vwp.n</code>)	<code>query</code> – coordinates in the format <code>lat0,lon0:lat1,lon1...</code>
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation.
<code>culture</code> (<code>c</code>)	<code>language</code> – See supported languages documentation.
<code>userRegion</code> (<code>ur</code>)	<code>view</code> – See supported views documentation.

The Azure Maps routing API also supports truck routing within the same API. The following table cross-references the additional Bing Maps truck routing parameters with the comparable API parameters in Azure Maps.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>dimensionUnit</code> (<code>dims</code>)	N/A – Dimensions in meters only supported.
<code>weightUnit</code> (<code>wu</code>)	N/A – Weights in kilograms only supported.
<code>vehicleHeight</code> (<code>height</code>)	<code>vehicleHeight</code>
<code>vehicleWidth</code> (<code>width</code>)	<code>vehicleWidth</code>
<code>vehicleLength</code> (<code>vl</code>)	<code>vehicleLength</code>
<code>vehicleWeight</code> (<code>weight</code>)	<code>vehicleWeight</code>
<code>vehicleAxles</code> (<code>axles</code>)	<code>vehicleAxelWeight</code>
<code>vehicleTrailers</code> (<code>vt</code>)	N/A
<code>vehicleSemi</code> (<code>semi</code>)	<code>vehicleCommercial</code>
<code>vehicleMaxGradient</code> (<code>vmg</code>)	N/A
<code>vehicleMinTurnRadius</code> (<code>vmtr</code>)	N/A
<code>vehicleAvoidCrossWind</code> (<code>vacw</code>)	N/A
<code>vehicleAvoidGroundingRisk</code> (<code>vagr</code>)	N/A
<code>vehicleHazardousMaterials</code> (<code>vhm</code>)	<code>vehicleLoadType</code>
<code>vehicleHazardousPermits</code> (<code>vhp</code>)	<code>vehicleLoadType</code>

TIP

By default, the Azure Maps route API only returns a summary (distance and times) and the coordinates for the route path. Use the `instructionsType` parameter to retrieve turn-by-turn instructions. The `routeRepresentation` parameter can be used to filter out the summary and route path.

Be sure to also review the [Best practices for routing](#) documentation.

The Azure Maps routing API has many additional features not available in Bing Maps that might be useful to integrate when migrating your app:

- Support for route type: shortest, fastest, trilling, and most fuel efficient.
- Support for additional travel modes: bicycle, bus, motorcycle, taxi, truck, and van.
- Support for 150 waypoints.
- Compute multiple travel times in a single request; historic traffic, live traffic, no traffic.
- Avoid additional road types: carpool roads, unpaved roads, already used roads.
- Engine specification-based routing. Calculate routes for combustion or electric vehicles based on their remaining fuel/charge and engine specifications.
- Specify maximum vehicle speed.

Snap coordinates to Road

There are several ways to snap coordinates to roads in Azure Maps.

- Use the route directions API to snap coordinates to a logical route along the road network.
- Use the Azure Maps Web SDK to snap individual coordinates to the nearest road in the vector tiles.
- Use the Azure Maps vector tiles directly to snap individual coordinates.

Using the route direction API to snap coordinates

Azure Maps can snap coordinates to roads by using the [route directions](#) API. This service can be used to reconstruct a logical route between a set of coordinates and is comparable to the Bing Maps Snap to Road API.

There are two different ways to use the route directions API to snap coordinates to roads.

- If there are 150 coordinates or less, they can be passed as waypoints in the GET route directions API. Using this approach two different types of snapped data can be retrieved; route instructions will contain the individual snapped waypoints, while the route path will have an interpolated set of coordinates that fill the full path between the coordinates.
- If there are more than 150 coordinates, the POST route directions API can be used. The coordinates start and end coordinates have to be passed into the query parameter, but all coordinates can be passed into the `supportingPoints` parameter in the body of the POST request and formatted a GeoJSON geometry collection of points. The only snapped data available using this approach will be the route path that is an interpolated set of coordinates that fill the full path between the coordinates. [Here is an example](#) of this approach using the services module in the Azure Maps Web SDK.

The following table cross-references the Bing Maps API parameters with the comparable API parameters in Azure Maps.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>points</code>	<code>supportingPoints</code> – pass these points into the body of the post request

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>interpolate</code>	N/A
<code>includeSpeedLimit</code>	N/A
<code>includeTruckSpeedLimit</code>	N/A
<code>speedUnit</code>	N/A
<code>travelMode</code>	<code>travelMode</code>
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation.
<code>culture (c)</code>	<code>language</code> – See supported languages documentation.
<code>userRegion (ur)</code>	<code>view</code> – See supported views documentation.

The Azure Maps routing API also supports truck routing parameter within the same API to ensure logical paths are calculated. The following table cross-references the additional Bing Maps truck routing parameters with the comparable API parameters in Azure Maps.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>dimensionUnit (dims)</code>	N/A – Dimensions in meters only supported.
<code>weightUnit (wu)</code>	N/A – Weights in kilograms only supported.
<code>vehicleHeight (height)</code>	<code>vehicleHeight</code>
<code>vehicleWidth (width)</code>	<code>vehicleWidth</code>
<code>vehicleLength (vl)</code>	<code>vehicleLength</code>
<code>vehicleWeight (weight)</code>	<code>vehicleWeight</code>
<code>vehicleAxles (axles)</code>	<code>vehicleAxelWeight</code>
<code>vehicleTrailers (vt)</code>	N/A
<code>vehicleSemi (semi)</code>	<code>vehicleCommercial</code>
<code>vehicleMaxGradient (vmg)</code>	N/A
<code>vehicleMinTurnRadius (vmtr)</code>	N/A
<code>vehicleAvoidCrossWind (vacw)</code>	N/A
<code>vehicleAvoidGroundingRisk (vagr)</code>	N/A

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
vehicleHazardousMaterials (vhm)	vehicleLoadType
vehicleHazardousPermits (vhp)	vehicleLoadType

Since this approach uses the route directions API, the full set of options in that service can be used to customize the logic used to snap the coordinate to roads. For example, specifying a departure time would result in historic traffic data being taken into consideration.

The Azure Maps route directions API does not currently return speed limit data, however that can be retrieved using the Azure Maps reverse geocoding API.

Using the Web SDK to snap coordinates

The Azure Maps Web SDK uses vector tiles to render the maps. These vector tiles contain the raw road geometry information and can be used to calculate the nearest road to a coordinate for simple snapping of individual coordinates. This is useful when you want the coordinates to visually appear over roads and you are already using the Azure Maps Web SDK to visualize the data.

This approach however will only snap to the road segments that are loaded within the map view. When zoomed out at country level there may be no road data, so snapping can't be done, however at that zoom level a single pixel can represent the area of several city blocks so snapping isn't needed. To address this, the snapping logic can be applied every time the map has finished moving. [Here is a code sample](#) that demonstrates this.

Using the Azure Maps vector tiles directly to snap coordinates

The Azure Maps vector tiles contain the raw road geometry data that can be used to calculate the nearest point on a road to a coordinate to do basic snapping of individual coordinates. All road segments appear in the sectors at zoom level 15, so you will want to retrieve tiles from there. You can then use the [quadtree tile pyramid math](#) to determine that tiles are needed and convert the tiles to geometries. From there a spatial math library, such as [turf.js](#) or [NetTopologySuite](#) can be used to calculate the closest line segments.

Retrieve a map image (Static Map)

Azure Maps provides an API for rendering the static map images with data overlaid. The Azure Maps [Map image render](#) API is comparable to the static map API in Bing Maps.

NOTE

Azure Maps requires the center, all pushpins and path locations to be coordinates in `longitude,latitude` format whereas Bing Maps uses the `latitude,longitude` format.

Addresses will need to be geocoded first.

The following table cross-references the Bing Maps API parameters with the comparable API parameters in Azure Maps.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
centerPoint	center
format	<code>format</code> – specified as part of URL path. Currently only PNG supported.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>heading</code>	N/A – Streetside not supported.
<code>imagerySet</code>	<code>layer</code> and <code>style</code> – See Supported map styles documentation.
<code>mapArea (ma)</code>	<code>bbox</code>
<code>mapLayer (ml)</code>	N/A
<code>mapSize (ms)</code>	<code>width</code> and <code>height</code> – can be up to 8192x8192 in size.
<code>declutterPins (dcl)</code>	N/A
<code>dpi</code>	N/A
<code>drawCurve</code>	<code>path</code>
<code>mapMetadata</code>	N/A
<code>pitch</code>	N/A – Streetside not supported.
<code>pushpin (pp)</code>	<code>pins</code>
<code>zoomLevel</code>	<code>zoom</code>
<code>query</code>	N/A – center or bounding box must be used.
<code>highlightEntity (he)</code>	N/A
<code>style</code>	N/A
route parameters	N/A
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation.
<code>culture (c)</code>	<code>language</code> – See supported languages documentation.
<code>userRegion (ur)</code>	<code>view</code> – See supported views documentation.

NOTE

Azure Maps uses a tile system with tiles that are twice the size of the map tiles used in Bing Maps. As such, the zoom level value in Azure Maps will appear one zoom level closer in Azure Maps compared to Bing Maps. Lower the zoom level in the requests you are migrating by 1 to compensate for this.

See the [How-to guide on the map image render API](#) for more information.

In addition to being able to generate a static map image, the Azure Maps render service also provides the ability

to directly access map tiles in raster (PNG) and vector format;

- [Map tile](#) – Retrieve raster (PNG) and vector tiles for the base maps (roads, boundaries, background).
- [Map imagery tile](#) – Retrieve aerial and satellite imagery tiles.

Pushpin URL parameter format comparison

Before: Bing Maps

In Bing Maps, pushpins can be added to a static map image by using the `pushpin` parameter in the URL. The `pushpin` parameter takes in a location in `latitude,longitude` format, an icon style and text label (up to three characters) as shown below:

```
&pushpin=latitude,longitude;iconStyle;label
```

Additional pushpins can be added by adding additional `pushpin` parameters to the URL with a different set of values. Pushpin icon styles are limited to one of the predefined styles available in the Bing Maps API.

For example, in Bing Maps, a red pushpin with the label "AB" can be added to the map at coordinates (longitude: -110, latitude: 45) with the following URL parameter:

```
&pushpin=45,-110;7;AB
```



After: Azure Maps

In Azure Maps, pushpins can also be added to a static map image by specifying the `pins` parameter in the URL. Pushpins in Azure Maps are defined by specifying an icon style and a list of locations that use that icon style. This information is then passed into the `pins` parameter. The `pins` parameter can be specified multiple times to support pushpins with different styles.

```
&pins=iconType|pinStyles||pinLocation1|pinLocation2|...
```

Additional styles can be used by adding additional `pins` parameters to the URL with a different style and set of locations.

When it comes to pin locations, Azure Maps requires the coordinates to be in `longitude latitude` format whereas Bing Maps uses `latitude,longitude` format. Also note that there is a space, not a comma separating longitude and latitude in Azure Maps.

The `iconType` value specifies the type of pin to create and can have the following values:

- `default` – The default pin icon.

- `none` – No icon is displayed, only labels will be rendered.
- `custom` – Specifies a custom icon is to be used. A URL pointing to the icon image can be added to the end of the `pins` parameter after the pin location information.
- `{udid}` – A Unique Data ID (UDID) for an icon stored in the Azure Maps Data Storage platform.

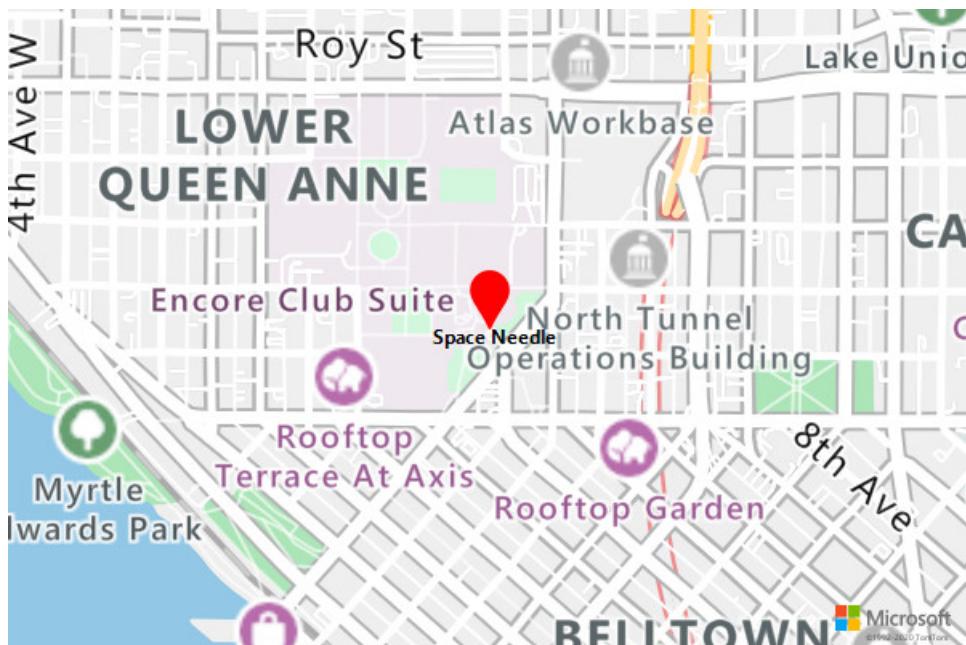
Pin styles in Azure Maps are added with the format `optionNameValue`, with multiple styles separated by pipe (`|`) characters like this `iconType|optionName1Value1|optionName2Value2`. Note the option names and values are not separated. The following style option names can be used to style pushpins in Azure Maps:

- `a1` – Specifies the opacity (alpha) of the pushpins. Can be a number between 0 and 1.
- `an` – Specifies the pin anchor. X and y pixel values specified in the format `x y`.
- `co` – The color of the pin. Must be a 24-bit hex color: `000000` to `FFFFFF`.
- `la` – Specifies the label anchor. X and y pixel values specified in the format `x y`.
- `lc` – The color of the label. Must be a 24-bit hex color: `000000` to `FFFFFF`.
- `ls` – The size of the label in pixels. Can be a number greater than 0.
- `ro` – A value in degrees to rotate the icon. Can be a number between -360 and 360.
- `sc` – A scale value for the pin icon. Can be a number greater than 0.

Label values are specified for each pin location rather than having a single label value that applies to all pushpins in the list of locations. The label value can be string of multiple characters and be wrapped with single quotes to ensure that it isn't mistaken as a style or location value.

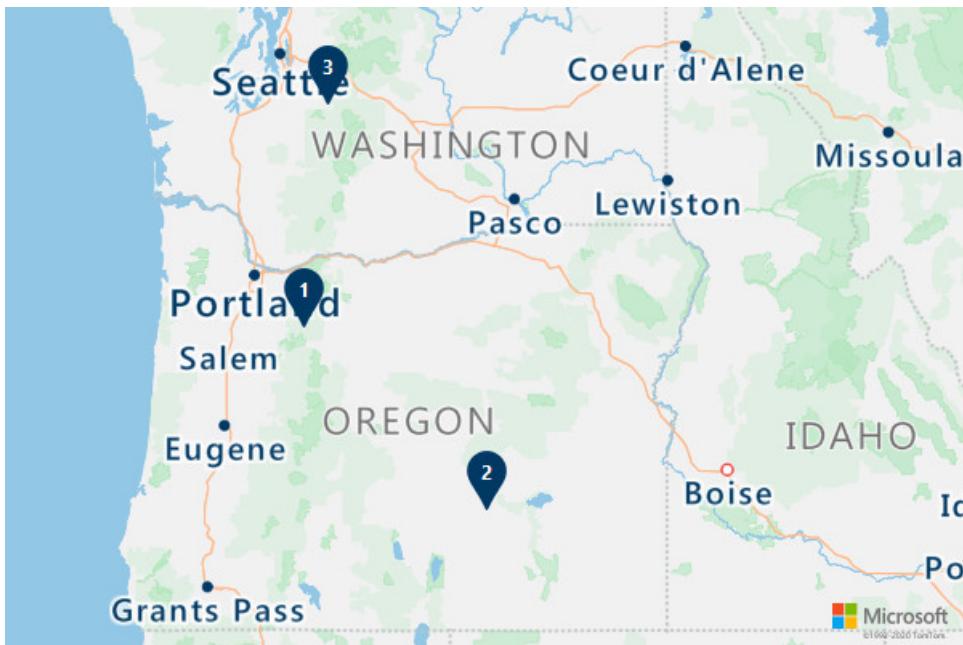
For example, in Azure Maps, adding a red (`FF0000`) default icon with the label "Space Needle" positioned below (15 50) the icon at coordinates (longitude: -122.349300, latitude: 47.620180) can be done with the following URL parameter:

```
&pins=default|coFF0000|la15 50||'Space Needle'-122.349300 47.620180
```



The following example adds three pins with the label values '1', '2', and '3':

```
&pins=default||'1'-122 45|'2'-119.5 43.2|'3'-121.67 47.12
```



Draw curve URL parameter format comparison

Before: Bing Maps

In Bing Maps, lines, and polygons can be added to a static map image by using the `drawCurve` parameter in the URL. The `drawCurve` parameter takes in a shape type, a style type and a list of locations to be rendered on the map as shown below:

```
&drawCurve=shapeType,styleType,location1,location2...
```

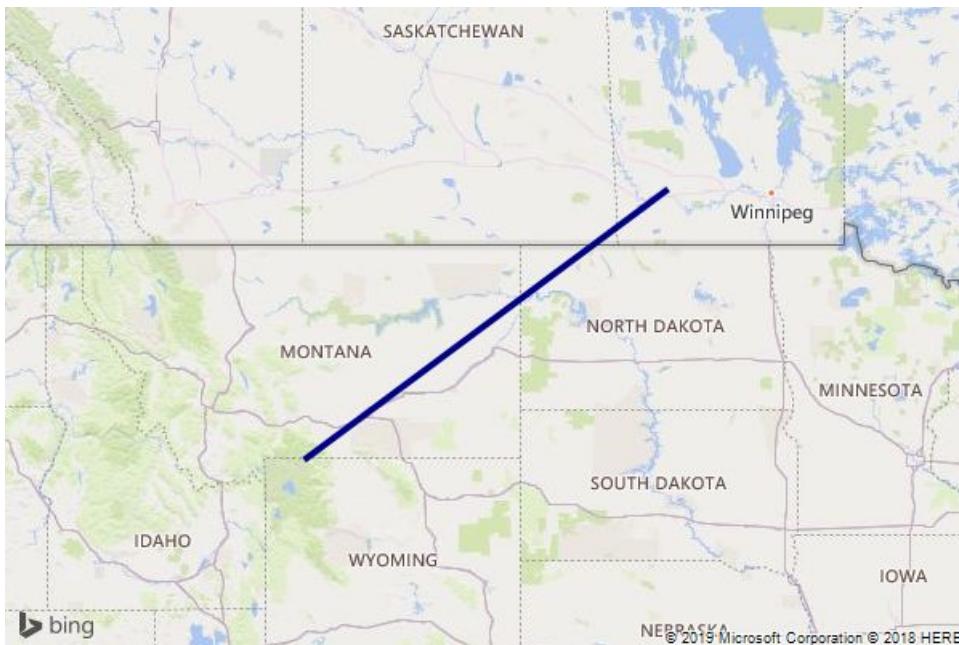
Additional styles can be used by adding additional `drawCurve` parameters to the URL with a different style and set of locations.

Locations in Bing Maps are specified with the format `latitude1,longitude1_latitude2,longitude2...`. Locations can also be encoded.

Shape types in Bing Maps include lines, polygons, circle, and curve. Style types include line color, line thickness, outline color, fill color, outline thickness, and circular radius.

For example, in Bing Maps, a blue line with 50% opacity and a thickness of four pixels can be added to the map between coordinates (longitude: -110, latitude: 45 and longitude: -100, latitude: 50) with the following URL parameter:

```
&drawCurve=1,FF000088,4;45,-110_50,-100
```



After: Azure Maps

In Azure Maps, lines and polygons can also be added to a static map image by specifying the `path` parameter in the URL. Like Bing Maps, a style and a list of locations can be specified in this parameter, and the `path` parameter can be specified multiple times to render multiple circles, lines and polygons with different styles.

```
&path=pathStyles||pathLocation1|pathLocation2|...
```

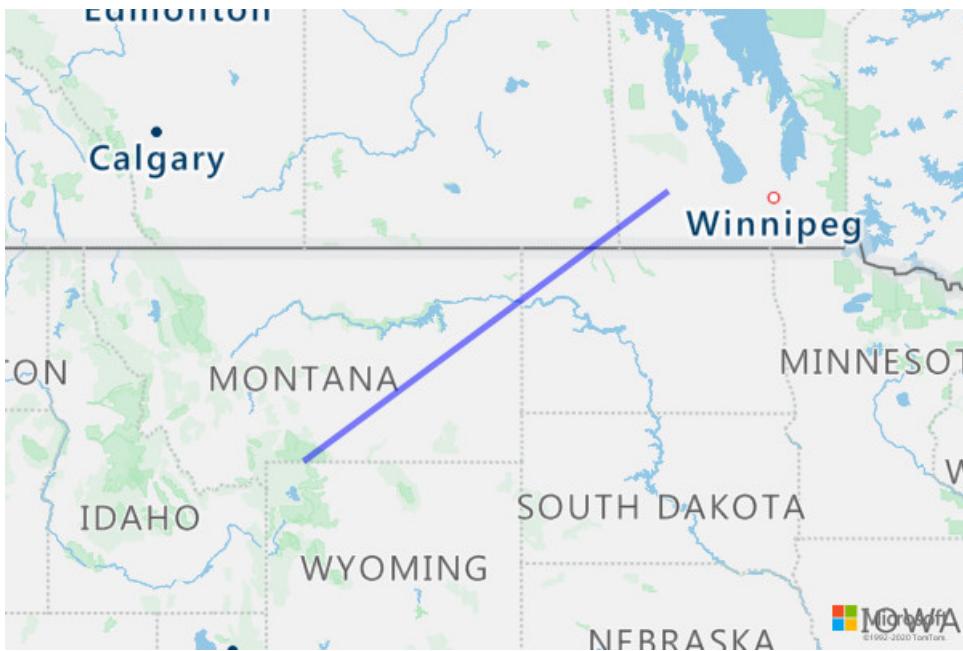
When it comes to path locations, Azure Maps requires the coordinates to be in `longitude latitude` format whereas Bing Maps uses `latitude,longitude` format. Also note that there is a space, not a comma separating longitude and latitude in Azure Maps. Azure Maps does not support encoded paths currently. Larger data sets can be uploaded as a GeoJSON fills into the Azure Maps Data Storage API as documented [here](#).

Path styles in Azure Maps are added with the format `optionNameValue`, with multiple styles separated by pipe (`|`) characters like this `optionName1Value1|optionName2Value2`. Note the option names and values are not separated. The following style option names can be used to style paths in Azure Maps:

- `fa` – The fill color opacity (alpha) used when rendering polygons. Can be a number between 0 and 1.
- `fc` – The fill color used to render the area of a polygon.
- `la` – The line color opacity (alpha) used when rendering lines and the outline of polygons. Can be a number between 0 and 1.
- `lc` – The line color used to render lines and the outline of polygons.
- `lw` – The width of the line in pixels.
- `ra` – Specifies a circles radius in meters.

For example, in Azure Maps, a blue line with 50% opacity and a thickness of four pixels can be added to the map between coordinates (longitude: -110, latitude: 45 and longitude: -100, latitude: 50) with the following URL parameter:

```
&path=lc0000FF|la.5|lw4||-110 45|-100 50
```



Calculate a distance matrix

Azure Maps provides an API for calculating the travel times and distances between a set of locations as a distance matrix. The Azure Maps distance matrix API is comparable to the distance matrix API in Bing Maps;

- **Route matrix:** Asynchronously calculates travel times and distances for a set of origins and destinations. Up to 700 cells per request is supported (the number of origins multiplied by the number of destinations). With that constraint in mind, examples of possible matrix dimensions are: `700x1` , `50x10` , `10x10` , `28x25` , `10x70` .

NOTE

A request to the distance matrix API can only be made using a POST request with the origin and destination information in the body of the request.

Additionally, Azure Maps requires all origins and destinations to be coordinates. Addresses will need to be geocoded first.

The following table cross-references the Bing Maps API parameters with the comparable API parameters in Azure Maps.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>origins</code>	<code>origins</code> – specify in the POST request body as GeoJSON.
<code>destinations</code>	<code>destination</code> – specify in the POST request body as GeoJSON.
<code>endTime</code>	<code>arriveAt</code>
<code>startTime</code>	<code>departAt</code>
<code>travelMode</code>	<code>travelMode</code>
<code>resolution</code>	N/A

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>distanceUnit</code>	N/A – All distances in meters.
<code>timeUnit</code>	N/A – All times in seconds.
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation.
<code>culture (c)</code>	<code>language</code> – See supported languages documentation.
<code>userRegion (ur)</code>	<code>view</code> – See supported views documentation.

TIP

All the advanced routing options available in the Azure Maps routing API (truck routing, engine specifications, avoid...) is support in the Azure Maps distance matrix API.

Calculate an isochrone

Azure Maps provides an API for calculating an isochrone, a polygon covering an area that can be traveled to in any direction from an origin point within a specified amount of time or amount of fuel/charge. The Azure Maps route range API is comparable to the isochrone API in Bing Maps;

- [Route Range](#)**: Calculate a polygon covering an area that can be traveled to in any direction from an origin point within a specified amount of time, distance, or amount of fuel/charge available.

NOTE

Azure Maps requires the query origin to be a coordinate. Addresses will need to be geocoded first.

Also, Bing Maps can calculate isochrones based on time or distance, while Azure Maps can calculate isochrones based on time, distance, or amount of fuel/charge available.

The following table cross-references the Bing Maps API parameters with the comparable API parameters in Azure Maps.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>waypoint (wp)</code>	<code>query</code>
<code>dateTime (dt)</code>	<code>departAt</code>
<code>maxTime</code>	<code>timeBudgetInSec</code>
<code>timeUnit (tu)</code>	N/A – All times in seconds.
<code>travelMode (mode)</code>	<code>travelMode</code>
<code>maxDistance (maxDis)</code>	<code>distanceBudgetInMeters</code>

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>distanceUnit (du)</code>	N/A – All distances in meters.
<code>optimize (optmz)</code>	<code>routeType</code>
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation.
<code>culture (c)</code>	<code>language</code> – See supported languages documentation.
<code>userRegion (ur)</code>	<code>view</code> – See supported views documentation.

TIP

All the advanced routing options available in the Azure Maps routing API (truck routing, engine specifications, avoid...) is support in the Azure Maps isochrone API.

Search for points of interest

Point of interest data can be searched in Bing Maps by using the following APIs:

- **Local search:** Searches for points of interest that are nearby (radial search), by name, or by entity type (category). The Azure Maps [POI search](#) and [POI category search](#) APIs are most like this API.
- **Location recognition:** Searches for points of interests that are within a certain distance of a location. The Azure Maps [nearby search](#) API is most like this API.
- **Local insights:** Searches for points of interests that are within a specified maximum driving time or distance from a specific coordinate. This is achievable with Azure Maps by first calculating an isochrone and then passing it into the [search within geometry](#) API.

Azure Maps provides several search APIs for points of interest:

- [POI search](#): Search for points of interests by name. For example; `"starbucks"`.
- [POI category search](#): Search for points of interests by category. For example; "restaurant".
- [Nearby search](#): Searches for points of interests that are within a certain distance of a location.
- [Fuzzy search](#): This API combines address geocoding with point of interest search. This API takes in a free-form string that can be an address, place, landmark, point of interest, or point of interest category and process the request immediately. This API is recommended for applications where users can search for addresses or points of interest from the same textbox.
- [Search within geometry](#): Search for points of interests that are within a specified geometry (polygon).
- [Search along route](#): Search for points of interests that are along a specified route path.
- [Fuzzy batch search](#): Create a request containing up to 10,000 addresses, places, landmarks, or point of interests and have them processed over a period of time. All the data will be processed in parallel on the server and when completed the full result set can be downloaded.

Be sure to review the [best practices for search](#) documentation.

Get traffic incidents

Azure Maps provides several APIs for retrieving traffic data. There are two types of traffic data available;

- **Flow data** – provides metrics on the flow of traffic on sections of roads. This is often used to color code

roads. This data is updated every 2 minutes.

- **Incident data** – provides data on construction, road closures, accidents, and other incidents that may affect traffic. This data is updated every minute.

Bing Maps provides traffic flow and incident data in its interactive map controls, and also make incident data available as a service.

Traffic data is also integrated into the Azure Maps interactive map controls. Azure maps also provides the following traffic services APIs;

- **Traffic flow segments**: Provides information about the speeds and travel times of the road fragment closest to the given coordinates.
- **Traffic flow tiles**: Provides raster and vector tiles containing traffic flow data. These can be used with the Azure Maps controls or in third-party map controls such as Leaflet. The vector tiles can also be used for advanced data analysis.
- **Traffic incident details**: Provides traffic incident details that are within a bounding box, zoom level, and traffic model.
- **Traffic incident tiles**: Provides raster and vector tiles containing traffic incident data.
- **Traffic incident viewport**: Retrieves the legal and technical information for the viewport described in the request, such as the traffic model ID.

The following table cross-references the Bing Maps traffic API parameters with the comparable traffic incident details API parameters in Azure Maps.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>mapArea</code>	<code>boundingBox</code> and <code>boundingZoom</code>
<code>includeLocationCodes</code>	N/A
<code>severity (s)</code>	N/A – all data returned
<code>type (t)</code>	N/A – all data returned
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation.
<code>culture (c)</code>	<code>language</code> – See supported languages documentation.
<code>userRegion (ur)</code>	<code>view</code> – See supported views documentation.

Get a time zone

Azure Maps provides an API for retrieving the time zone a coordinate is in. The Azure Maps time zone API is comparable to the time zone API in Bing Maps;

- **Time zone by coordinate**: Specify a coordinate and get the details for the time zone it falls in.

The following table cross-references the Bing Maps API parameters with the comparable API parameters in Azure Maps.

BING MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
point	query
query	N/A - locations must be geocoded first.
dateTime	timeStamp
includeDstRules	N/A – Always included in response by Azure Maps.
key	subscription-key – See also the Authentication with Azure Maps documentation.
culture (c)	language – See supported languages documentation.
userRegion (ur)	view – See supported views documentation.

In addition to this the Azure Maps platform also provides a number of additional time zone APIs to help with conversions with time zone names and IDs;

- [Time zone by ID](#): Returns current, historical, and future time zone information for the specified IANA time zone ID.
- [Time zone Enum IANA](#): Returns a full list of IANA time zone IDs. Updates to the IANA service are reflected in the system within one day.
- [Time zone Enum Windows](#): Returns a full list of Windows Time Zone IDs.
- [Time zone IANA version](#): Returns the current IANA version number used by Azure Maps.
- [Time zone Windows to IANA](#): Returns a corresponding IANA ID, given a valid Windows Time Zone ID.
Multiple IANA IDs may be returned for a single Windows ID.

Spatial Data Services (SDS)

The spatial data services in Bing Maps provide three key functionalities:

- Batch geocoding – Process a large batch of address geocodes with a single request.
- Retrieve administrative boundary data – Use a coordinate and get an intersecting boundary for a specified entity type.
- Host and query spatial business data – Upload a simple 2D table of data and access it using a few simple spatial queries.

Batch geocode data

Batch geocoding is the process of taking a large number of addresses or places, passing them all in a single request to a service, and having all of them geocoded in parallel and the results returned in a single response.

Bing Maps allows up to 200,000 addresses to be passed in a single batch geocode request. This request goes into a queue and usually processes over a period of time, anywhere from a few minutes to a few hours depending on the size of the data set and the load on the service. Each address in the request generated a transaction.

Azure Maps has a batch geocoding service, however it allows up to 10,000 addresses to be passed in a single request and is processed over seconds to a few minutes depending on the size of the data set and the load on the service. Each address in the request generated a transaction. In Azure Maps, the batch geocoding service is only available the Gen 2 or S1 pricing tier. For more information on pricing tiers, see [Choose the right pricing tier in Azure Maps](#).

Another option for geocoding a large number addresses with Azure Maps is to make parallel requests to the standard search APIs. These services only accept a single address per request but can be used with the S0 tier that also provides free usage limits. The S0 tier allows up to 50 requests per second to the Azure Maps platform from a single account. So if you process limit these to stay within that limit, it is possible to geocode upwards of 180,000 address an hour. The Gen 2 or S1 pricing tier doesn't have a documented limit on the number of queries per second that can be made from an account, so a lot more data can be processed faster when using that pricing tier, however using the batch geocoding service will help reduce the total amount of data transferred and will drastically reduce the network traffic.

- [Free-form address geocoding](#): Specify a single address string (like "1 Microsoft way, Redmond, WA") and process the request immediately. This service is recommended if you need to geocode individual addresses quickly.
- [Structured address geocoding](#): Specify the parts of a single address, such as the street name, city, country, and postal code and process the request immediately. This service is recommended if you need to geocode individual addresses quickly and the data is already parsed into its individual address parts.
- [Batch address geocoding](#): Create a request containing up to 10,000 addresses and have them processed over a period of time. All the addresses will be geocoded in parallel on the server and when completed the full result set can be downloaded. This service is recommended for geocoding large data sets.
- [Fuzzy search](#): This API combines address geocoding with point of interest search. This API takes in a free-form string that can be an address, place, landmark, point of interest, or point of interest category and process the request immediately. This API is recommended for applications where users can search for addresses or points of interest from the same textbox.
- [Fuzzy batch search](#): Create a request containing up to 10,000 addresses, places, landmarks, or point of interests and have them processed over a period of time. All the data will be processed in parallel on the server and when completed the full result set can be downloaded.

Get administrative boundary data

In Bing Maps, administrative boundaries for countries, states, counties, cities, and postal codes are made available via the Geodata API. This API takes in either a coordinate or query to geocode. If a query is passed in, it is geocoded and the coordinates from the first result is used. This API takes the coordinates and retrieves the boundary of the specified entity type that intersects the coordinate. Note that this API did not necessarily return the boundary for the query that was passed in. If a query for "Seattle, WA" is passed in, but the entity type value is set to country region, the boundary for the USA would be returned.

Azure Maps also provides access to administrative boundaries (countries, states, counties, cities, and postal codes). To retrieve a boundary, you must query one of the search APIs for the boundary you want (i.e. Seattle, WA). If the search result has an associated boundary, a geometry ID will be provided in the result response. The search polygon API can then be used to retrieve the exact boundaries for one or more geometry IDs. This is a bit different than Bing Maps as Azure Maps returns the boundary for what was searched for, whereas Bing Maps returns a boundary for a specified entity type at a specified coordinate. Additionally, the boundary data returned by Azure Maps is in GeoJSON format.

To recap:

1. Pass a query for the boundary you want to receive into one of the following search APIs.
 - [Free-form address geocoding](#)
 - [Structured address geocoding](#)
 - [Batch address geocoding](#)
 - [Fuzzy search](#)
 - [Fuzzy batch search](#)
2. If the desired result(s) has a geometry ID(s), pass it into the [Search Polygon API](#).

Host and query spatial business data

The spatial data services in Bing Maps provide a simple spatial data storage solution for hosting business data and exposing it as a spatial REST service. This service provides four main queries; find by property, find nearby, find in bounding box, and find with 1 mile of a route. Many companies who use this service, often already have their business data already stored in a database somewhere and have been uploading a small subset of it into this service to power applications like store locators. Since key-based authentication provides basic security, it has been recommended that this service only be used with public facing data.

Most business location data starts off in a database. As such it is recommended to use existing Azure storage solutions such as Azure SQL or Azure PostgreSQL (with the PostGIS plugin). Both of these storage solutions support spatial data and provide a rich set of spatial querying capabilities. Once your data is in a suitable storage solution, it can then be integrated into your application by creating a custom web service, or by using a framework such as ASP.NET or Entity Framework. Using this approach provides more querying capabilities and as well as much higher security options.

Azure Cosmos DB also provides a limited set of spatial capabilities that, depending on your scenario, may be sufficient.

Here are some useful resources around hosting and querying spatial data in Azure.

- [Azure SQL Spatial Data Types overview](#)
- [Azure SQL Spatial – Query nearest neighbor](#)
- [Azure Cosmos DB geospatial capabilities overview](#)

Client libraries

Azure Maps provides client libraries for the following programming languages;

- JavaScript, TypeScript, Node.js – [documentation | NPM package](#)

Open-source client libraries for other programming languages;

- .NET Standard 2.0 – [GitHub project | NuGet package](#)

Clean up resources

No resources to be cleaned up.

Next steps

Learn more about the Azure Maps REST services.

[Best practices for using the search service](#)

Tutorial: Migrate from Google Maps to Azure Maps

3/5/2021 • 4 minutes to read • [Edit Online](#)

This article provides insights on how to migrate web, mobile and server-based applications from Google Maps to the Microsoft Azure Maps platform. This tutorial includes comparative code samples, migration suggestions, and best practices for migrating to Azure Maps. In this tutorial, you will learn:

- High-level comparison for equivalent Google Maps features available in Azure Maps.
- What licensing differences to take into consideration.
- How to plan your migration.
- Where to find technical resources and support.

Prerequisites

1. Sign in to the [Azure portal](#). If you don't have an Azure subscription, create a [free account](#) before you begin.
2. [Make an Azure Maps account](#)
3. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

Azure Maps platform overview

Azure Maps provides developers from all industries powerful geospatial capabilities. The capabilities are packed with regularly updated map data to provide geographic context for web, and mobile applications. Azure Maps has an Azure One API compliant set of REST APIs. The REST APIs offer Maps Rendering, Search, Routing, Traffic, Time Zones, Geolocation, Geofencing, Map Data, Weather, Mobility, and Spatial Operations. Operations are accompanied by both Web and Android SDKs to make development easy, flexible, and portable across multiple platforms.

High-level platform comparison

The table provides a high-level list of Azure Maps features, which correspond to Google Maps features. This list doesn't show all of the Azure Maps features. Additional Azure Maps features include: accessibility, geofencing, isochrones, spatial operations, direct map tile access, batch services, and data coverage comparisons (that is, imagery coverage).

GOOGLE MAPS FEATURE	AZURE MAPS SUPPORT
Web SDK	✓
Android SDK	✓
iOS SDK	Planned
REST Service APIs	✓
Directions (Routing)	✓
Distance Matrix	✓

GOOGLE MAPS FEATURE	AZURE MAPS SUPPORT
Elevation	✓ (Preview)
Geocoding (Forward/Reverse)	✓
Geolocation	N/A
Nearest Roads	✓
Places Search	✓
Places Details	N/A – website & phone number available
Places Photos	N/A
Place Autocomplete	✓
Snap to Road	✓
Speed Limits	✓
Static Maps	✓
Static Street View	N/A
Time Zone	✓
Maps Embedded API	N/A
Map URLs	N/A

Google Maps provides basic key-based authentication. Azure Maps provides both basic key-based authentication and Azure Active Directory authentication. Azure Active Directory authentication provides more security features, compared to the basic key-based authentication.

Licensing considerations

When migrating to Azure Maps from Google Maps, consider the following points about licensing.

- Azure Maps charges for the usage of interactive maps, which is based on the number of loaded map tiles. On the other hand, Google Maps charges for loading the map control. In the interactive Azure Maps SDKs, map tiles are automatically cached to reduce the development cost. One Azure Maps transaction is generated for every 15 map tiles that are loaded. The interactive Azure Maps SDKs uses 512-pixel tiles, and on average, it generates one or less transactions per page view.
- Often, its more cost effective to replace static map images from Google Maps web services with the Azure Maps Web SDK. The Azure Maps Web SDK uses map tiles. Unless the user pans and zooms the map, the service often generates only a fraction of a transaction per map load. The Azure Maps web SDK has options for disabling panning and zooming, if desired. Additionally, the Azure Maps web SDK provides a lot more visualization options than the static map web service.
- Azure Maps allows data from its platform to be stored in Azure. Also, data can be cached elsewhere for up to six months as per the [terms of use](#).

Here are some related resources for Azure Maps:

- [Azure Maps pricing page](#)
- [Azure pricing calculator](#)
- [Azure Maps term of use](#) (included in the Microsoft Online Services Terms)
- [Choose the right pricing tier in Azure Maps](#)

Suggested migration plan

The following is a high-level migration plan.

1. Take inventory of the Google Maps SDKs and services that your application uses. Verify that Azure Maps provides alternative SDKs and services.
2. If you don't already have one, create an Azure subscription at <https://azure.com>.
3. Create an Azure Maps account ([documentation](#)) and authentication key or Azure Active Directory ([documentation](#)).
4. Migrate your application code.
5. Test your migrated application.
6. Deploy your migrated application to production.

Create an Azure Maps account

To create an Azure Maps account and get access to the Azure Maps platform, follow these steps:

1. If you don't have an Azure subscription, create a [free account](#) before you begin.
2. Sign in to the [Azure portal](#).
3. Create an [Azure Maps account](#).
4. [Get your Azure Maps subscription key](#) or setup Azure Active Directory authentication for enhanced security.

Azure Maps technical resources

Here is a list of useful technical resources for Azure Maps.

- Overview: <https://azure.com/maps>
- Documentation: <https://aka.ms/AzureMapsDocs>
- Web SDK Code Samples: <https://aka.ms/AzureMapsSamples>
- Developer Forums: <https://aka.ms/AzureMapsForums>
- Videos: <https://aka.ms/AzureMapsVideos>
- Blog: <https://aka.ms/AzureMapsBlog>
- Tech Blog: <https://aka.ms/AzureMapsTechBlog>
- Azure Maps Feedback (UserVoice): <https://aka.ms/AzureMapsFeedback>
- [Azure Maps Jupyter Notebook](#)

Migration support

Developers can seek migration support through the [forums](#) or through one of the many Azure support options:
<https://azure.microsoft.com/support/options>

Clean up resources

No resources to be cleaned up.

Next steps

Learn the details of how to migrate your Google Maps application with these articles:

[Migrate a web app](#)

Tutorial: Migrate a web app from Google Maps

3/5/2021 • 42 minutes to read • [Edit Online](#)

Most web apps, which use Google Maps, are using the Google Maps V3 JavaScript SDK. The Azure Maps Web SDK is the suitable Azure-based SDK to migrate to. The Azure Maps Web SDK lets you customize interactive maps with your own content and imagery. You can run your app on both web or mobile applications. This control makes use of WebGL, allowing you to render large data sets with high performance. Develop with this SDK using JavaScript or TypeScript. In this tutorial, you will learn how to:

- Load a map
- Localize a map
- Add markers, polylines, and polygons.
- Display information in a popup or info window
- Load and display KML and GeoJSON data
- Cluster markers
- Overlay a tile layer
- Show traffic data
- Add a ground overlay

You will also learn:

- How to accomplish common mapping tasks using the Azure Maps Web SDK.
- Best practices to improve performance and user experience.
- Tips on how to make your application using more advance features available in Azure Maps.

If migrating an existing web application, check to see if it is using an open-source map control library. Examples of open-source map control library are: Cesium, Leaflet, and OpenLayers. You can still migrate your application, even if it uses an open-source map control library, and you do not want to use the Azure Maps Web SDK. In such case, connect your application to the Azure Maps tile services ([road tiles](#) | [satellite tiles](#)). The following points detail on how to use Azure Maps in some commonly used open-source map control libraries.

- Cesium - A 3D map control for the web. [Code sample](#) | [Documentation](#)
- Leaflet – Lightweight 2D map control for the web. [Code sample](#) | [Documentation](#)
- OpenLayers - A 2D map control for the web that supports projections. [Code sample](#) | [Documentation](#)

If developing using a JavaScript framework, one of the following open-source projects may be useful:

- [ng-azure-maps](#) - Angular 10 wrapper around Azure maps.
- [AzureMapsControl.Components](#) - An Azure Maps Blazor component.
- [Azure Maps React Component](#) - A react wrapper for the Azure Maps control.
- [Vue Azure Maps](#) - An Azure Maps component for Vue application.

Prerequisites

1. Sign in to the [Azure portal](#). If you don't have an Azure subscription, create a [free account](#) before you begin.
2. [Make an Azure Maps account](#)
3. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

Key features support

The table lists key API features in the Google Maps V3 JavaScript SDK and the supported API feature in the Azure Maps Web SDK.

GOOGLE MAPS FEATURE	AZURE MAPS WEB SDK SUPPORT
Markers	✓
Marker clustering	✓
Polylines & Polygons	✓
Data layers	✓
Ground Overlays	✓
Heat maps	✓
Tile Layers	✓
KML Layer	✓
Drawing tools	✓
Geocoder service	✓
Directions service	✓
Distance Matrix service	✓
Elevation service	✓

Notable differences in the web SDKs

The following are some key differences between the Google Maps and Azure Maps Web SDKs, to be aware of:

- In addition to providing a hosted endpoint for accessing the Azure Maps Web SDK, an NPM package is available. Embed the Web SDK package into apps. For more information, see this [documentation](#). This package also includes TypeScript definitions.
- You first need to create an instance of the Map class in Azure Maps. Wait for the maps `ready` or `load` event to fire before programmatically interacting with the map. This order will ensure that all the map resources have been loaded and are ready to be accessed.
- Both platforms use a similar tiling system for the base maps. The tiles in Google Maps are 256 pixels in dimension; however, the tiles in Azure Maps are 512 pixels in dimension. To get the same map view in Azure Maps as Google Maps, subtract Google Maps zoom level by the number one in Azure Maps.
- Coordinates in Google Maps are referred to as `[latitude, longitude]`, while Azure Maps uses `[longitude, latitude]`. The Azure Maps format is aligned with the standard `[x, y]`, which is followed by most GIS platforms.
- Shapes in the Azure Maps Web SDK are based on the GeoJSON schema. Helper classes are exposed through the [atlas.data namespace](#). There's also the [atlas.Shape](#) class. Use this class to wrap GeoJSON objects, to make it easy to update and maintain the data bindable way.

- Coordinates in Azure Maps are defined as Position objects. A coordinate is specified as a number array in the format `[longitude,latitude]`. Or, it's specified using new `atlas.data.Position(longitude, latitude)`.

TIP

The Position class has a static helper method for importing coordinates that are in "latitude, longitude" format. The `atlas.data.Position.fromLatLng` method can often be replaced with the `new google.maps.LatLng` method in Google Maps code.

- Rather than specifying styling information on each shape that is added to the map, Azure Maps separates styles from the data. Data is stored in a data source, and is connected to rendering layers. Azure Maps code uses data sources to render the data. This approach provides enhanced performance benefit. Additionally, many layers support data-driven styling where business logic can be added to layer style options. This support changes how individual shapes are rendered within a layer based on properties defined in the shape.

Web SDK side-by-side examples

This collection has code samples for each platform, and each sample covers a common use case. It's intended to help you migrate your web application from Google Maps V3 JavaScript SDK to the Azure Maps Web SDK. Code samples related to web applications are provided in JavaScript. However, Azure Maps also provides TypeScript definitions as an additional option through an [NPM module](#).

Topics

- [Load a map](#)
- [Localizing the map](#)
- [Setting the map view](#)
- [Adding a marker](#)
- [Adding a custom marker](#)
- [Adding a polyline](#)
- [Adding a polygon](#)
- [Display an info window](#)
- [Import a GeoJSON file*](#)
- [Marker clustering](#)
- [Add a heat map](#)
- [Overlay a tile layer](#)
- [Show traffic data](#)
- [Add a ground overlay](#)
- [Add KML data to the map](#)

Load a map

Both SDKs have the same steps to load a map:

- Add a reference to the Map SDK.
- Add a `div` tag to the body of the page, which will act as a placeholder for the map.
- Create a JavaScript function that gets called when the page has loaded.
- Create an instance of the respective map class.

Some key differences

- Google maps requires an account key to be specified in the script reference of the API. Authentication credentials for Azure Maps are specified as options of the map class. This credential can be a subscription key

or Azure Active Directory information.

- Google Maps accepts a callback function in the script reference of the API, which is used to call an initialization function to load the map. With Azure Maps, the `onload` event of the page should be used.
- When referencing the `div` element in which the map will be rendered, the `Map` class in Azure Maps only requires the `id` value while Google Maps requires a `HTMLElement` object.
- Coordinates in Azure Maps are defined as Position objects, which can be specified as a simple number array in the format `[longitude, latitude]`.
- The zoom level in Azure Maps is one level lower than the zoom level in Google Maps. This discrepancy is because the difference in the sizes of tiling system of the two platforms.
- Azure Maps doesn't add any navigation controls to the map canvas. So, by default, a map doesn't have zoom buttons and map style buttons. But, there are control options for adding a map style picker, zoom buttons, compass or rotation control, and a pitch control.
- An event handler is added in Azure Maps to monitor the `ready` event of the map instance. This event will fire when the map has finished loading the WebGL context and all the needed resources. Add any code you want to run after the map completes loading, to this event handler.

The basic examples below uses Google Maps to load a map centered over New York at coordinates. The longitude: -73.985, latitude: 40.747, and the map is at zoom level of 12.

Before: Google Maps

Display a Google Map centered and zoomed over a location.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <script type='text/javascript'>
        var map;

        function initMap() {
            map = new google.maps.Map(document.getElementById('myMap'), {
                center: new google.maps.LatLng(40.747, -73.985),
                zoom: 12
            });
        }
    </script>

    <!-- Google Maps Script Reference -->
    <script src="https://maps.googleapis.com/maps/api/js?callback=initMap&key=[Your Google Maps Key]" async defer></script>
</head>
<body>
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>
```

Running this code in a browser will display a map that looks like the following image:



After: Azure Maps

Load a map with the same view in Azure Maps along with a map style control and zoom buttons.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css"
type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <script type='text/javascript'>
        var map;

        function initMap() {
            map = new atlas.Map('myMap', {
                center: [-73.985, 40.747], //Format coordinates as longitude, latitude.
                zoom: 11, //Subtract the zoom level by one.

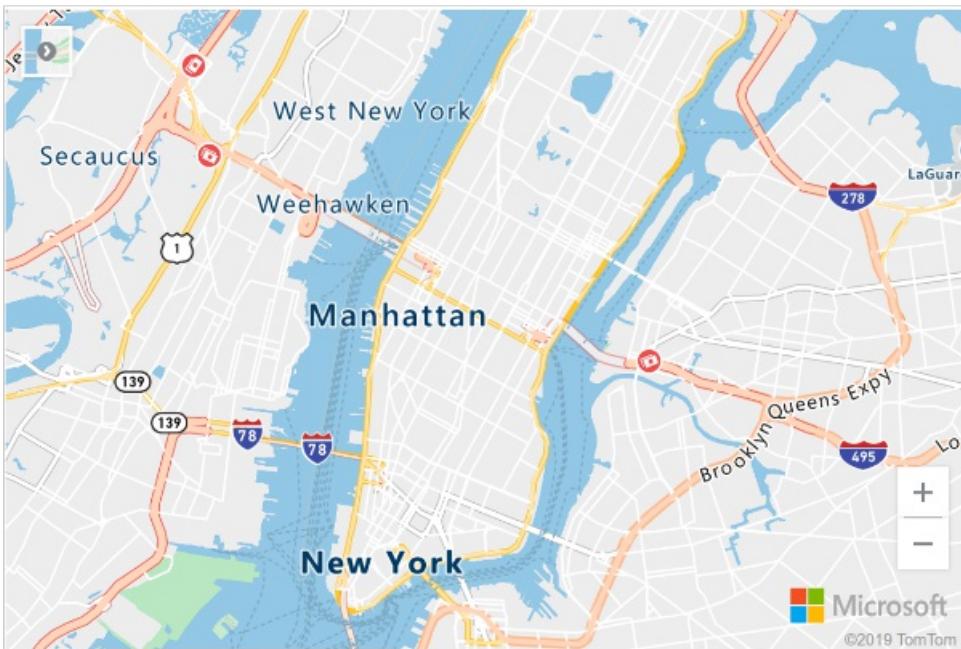
                //Specify authentication information when loading the map.
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

            //Wait until the map resources are ready.
            map.events.add('ready', function () {
                //Add zoom controls to bottom right of map.
                map.controls.add(new atlas.control.ZoomControl(), {
                    position: 'bottom-right'
                });

                //Add map style control in top left corner of map.
                map.controls.add(new atlas.control.StyleControl(), {
                    position: 'top-left'
                });
            });
        }
    </script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```

Running this code in a browser will display a map that looks like the following image:



Find detailed documentation on how to set up and use the Azure Maps map control in a web app, by clicking [here](#).

NOTE

Unlike Google Maps, Azure Maps does not require an initial center and a zoom level to load the map. If this information is not provided when loading the map, Azure maps will try to determine city of the user. It will center and zoom the map there.

Additional resources:

- Azure Maps also provides navigation controls for rotating and pitching the map view, as documented [here](#).

Localizing the map

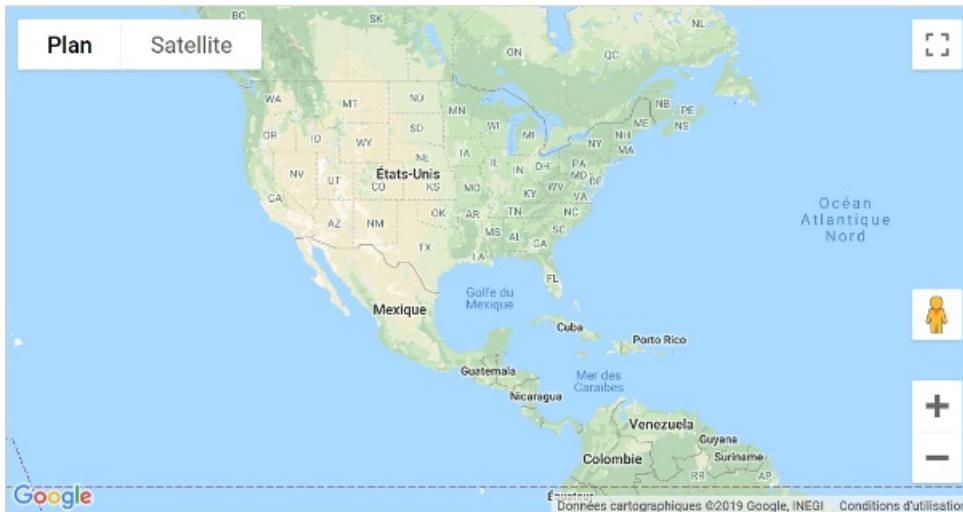
If your audience is spread across multiple countries/regions or speak different languages, localization is important.

Before: Google Maps

To localize Google Maps, add language and region parameters.

```
<script type="text/javascript" src=" https://maps.googleapis.com/maps/api/js?callback=initMap&key=[api_key]&language=[language_code]&region=[region_code]" async defer></script>
```

Here is an example of Google Maps with the language set to "fr-FR".



After: Azure Maps

Azure Maps provides two different ways of setting the language and regional view of the map. The first option is to add this information to the global *atlas* namespace. It will result in all map control instances in your app defaulting to these settings. The following sets the language to French ("fr-FR") and the regional view to "Auto":

```
atlas.setLanguage('fr-FR');
atlas.setView('auto');
```

The second option is to pass this information into the map options when loading the map. Like this:

```
map = new atlas.Map('myMap', {
    language: 'fr-FR',
    view: 'auto',

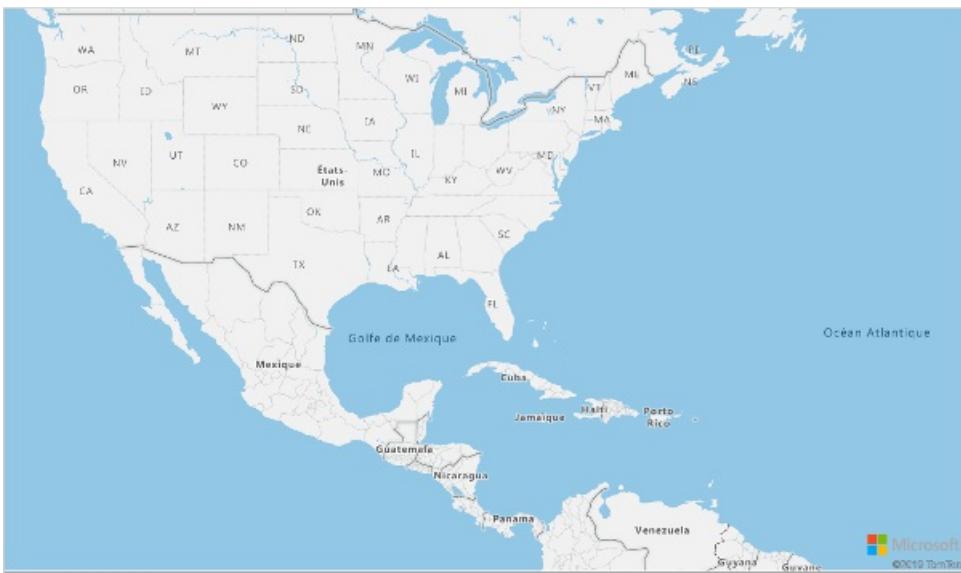
    authOptions: {
        authType: 'subscriptionKey',
        subscriptionKey: '<Your Azure Maps Key>'
    }
});
```

NOTE

With Azure Maps, it is possible to load multiple map instances on the same page with different language and region settings. It is also possible to update these settings in the map after it has loaded.

Find a detailed list of [supported languages](#) in Azure Maps.

Here is an example of Azure Maps with the language set to "fr" and the user region set to "fr-FR".



Setting the map view

Dynamic maps in both Azure and Google Maps can be programmatically moved to new geographic locations. To do so, call the appropriate functions in JavaScript. The examples show how to make the map display satellite aerial imagery, center the map over a location, and change the zoom level to 15 in Google Maps. The following location coordinates are used: longitude: -111.0225 and latitude: 35.0272.

NOTE

Google Maps uses tiles that are 256 pixels in dimensions, while Azure Maps uses a larger 512-pixel tile. Thus, Azure Maps requires less number of network requests to load the same map area as Google Maps. Due to the way tile pyramids work in map controls, you need to subtract the zoom level used in Google Maps by the number one when using Azure Maps. This arithmetic operation ensures that larger tiles in Azure Maps render that same map area as in Google Maps,

Before: Google Maps

Move the Google Maps map control using the `setOptions` method. This method allows you to specify the center of the map and a zoom level.

```
map.setOptions({
  mapTypeId: google.maps.MapTypeId.SATELLITE,
  center: new google.maps.LatLng(35.0272, -111.0225),
  zoom: 15
});
```



After: Azure Maps

In Azure Maps, change the map position using the `setCamera` method and change the map style using the `setStyle` method. The coordinates in Azure Maps are in "longitude, latitude" format, and the zoom level value is subtracted by one.

```
map.setCamera({
    center: [-111.0225, 35.0272],
    zoom: 14
});

map.setStyle({
    style: 'satellite'
});
```



Additional resources:

- [Choose a map style](#)
- [Supported map styles](#)

Adding a marker

In Azure Maps, there are multiple ways in which point data can be rendered on the map:

- **HTML Markers** – Renders points using traditional DOM elements. HTML Markers support dragging.
- **Symbol Layer** – Renders points with an icon or text within the WebGL context.
- **Bubble Layer** – Renders points as circles on the map. The radii of the circles can be scaled based on properties in the data.

Render Symbol layers and Bubble layers within the WebGL context. Both layers can render large sets of points on the map. These layers require data to be stored in a data source. Data sources and rendering layers should be added to the map after the `ready` event has fired. HTML Markers are rendered as DOM elements within the page, and they don't use a data source. The more DOM elements a page has, the slower the page becomes. If rendering more than a few hundred points on a map, it's recommended to use one of the rendering layers instead.

Let's add a marker to the map at with the number 10 overlaid as a label. Use longitude: -0.2 and latitude: 51.5.

Before: Google Maps

With Google Maps, add markers to the map using the `google.maps.Marker` class and specify the map as one of the options.

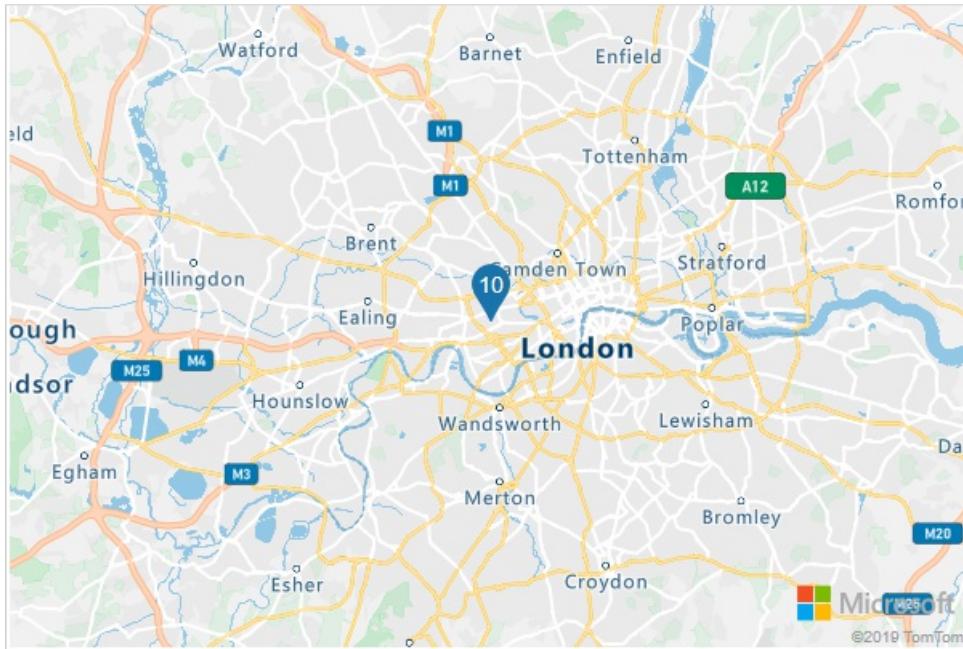
```
//Create a marker and add it to the map.  
var marker = new google.maps.Marker({  
    position: new google.maps.LatLng(51.5, -0.2),  
    label: '10',  
    map: map  
});
```



After: Azure Maps using HTML Markers

In Azure Maps, use HTML markers to display a point on the map. HTML markers are recommended for apps that only need to display a small number of points on the map. To use an HTML marker, create an instance of the `atlas.HtmlMarker` class. Set the text and position options, and add the marker to the map using the `map.markers.add` method.

```
//Create a HTML marker and add it to the map.  
map.markers.add(new atlas.HtmlMarker({  
    text: '10',  
    position: [-0.2, 51.5]  
}));
```



After: Azure Maps using a Symbol Layer

For a Symbol layer, add the data to a data source. Attach the data source to the layer. Additionally, the data source and layer should be added to the map after the `ready` event has fired. To render a unique text value above a symbol, the text information needs to be stored as a property of the data point. The property must be referenced in the `textField` option of the layer. This approach is a bit more work than using HTML markers, but it better performance.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <script type='text/javascript'>
        var map, datasource;

        function initMap() {
            map = new atlas.Map('myMap', {
                center: [-0.2, 51.5],
                zoom: 9,

                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

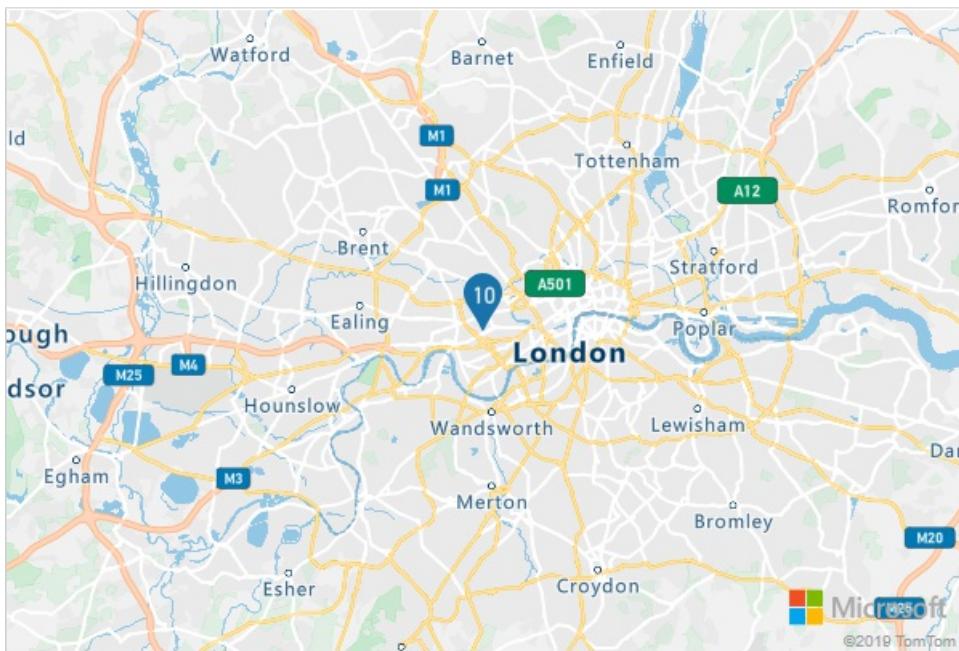
            //Wait until the map resources are ready.
            map.events.add('ready', function () {

                //Create a data source and add it to the map.
                datasource = new atlas.source.DataSource();
                map.sources.add(datasource);

                //Create a point feature, add a property to store a label for it, and add it to the data source.
                datasource.add(new atlas.data.Feature(new atlas.data.Point([-0.2, 51.5]), {
                    label: '10'
                }));

                //Add a layer for rendering point data as symbols.
                map.layers.add(new atlas.layer.SymbolLayer(datasource, null, {
                    textOptions: {
                        //Use the label property to populate the text for the symbols.
                        textField: ['get', 'label'],
                        color: 'white',
                        offset: [0, -1]
                    }
                }));
            });
        }
    </script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



Additional resources:

- [Create a data source](#)
- [Add a Symbol layer](#)
- [Add a Bubble layer](#)
- [Cluster point data](#)
- [Add HTML Markers](#)
- [Use data-driven style expressions](#)
- [Symbol layer icon options](#)
- [Symbol layer text option](#)
- [HTML marker class](#)
- [HTML marker options](#)

Adding a custom marker

You may use Custom images to represent points on a map. The map below uses a custom image to display a point on the map. The point is displayed at latitude: 51.5 and longitude: -0.2. The anchor offsets the position of the marker, so that the point of the pushpin icon aligns with the correct position on the map.

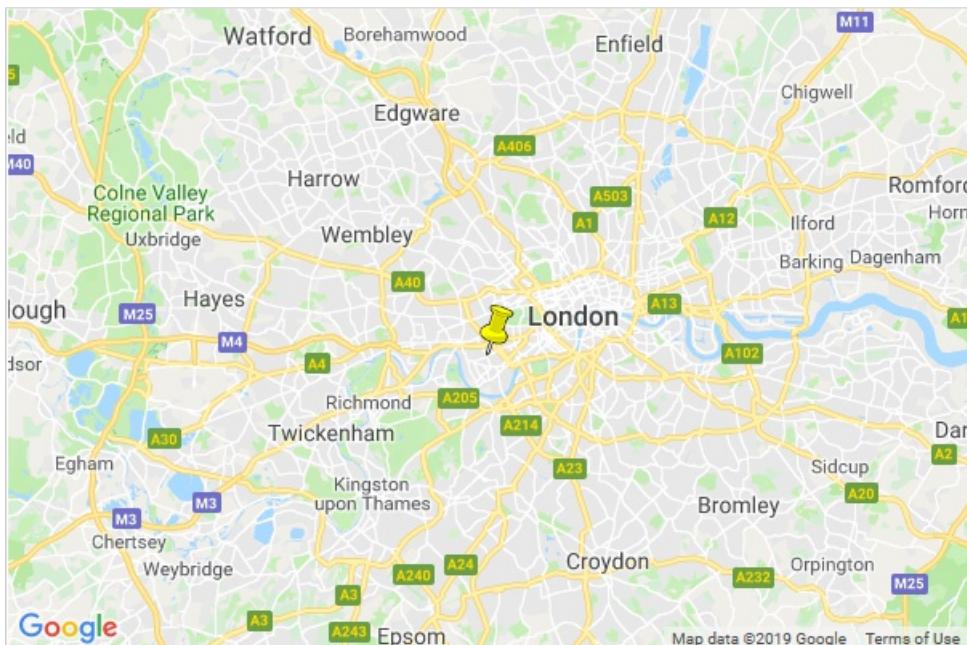


yellow-pushpin.png

Before: Google Maps

Create a custom marker by specifying an `Icon` object that contains the `url` to the image. Specify an `anchor` point to align the point of the pushpin image with the coordinate on the map. The anchor value in Google Maps is relative to the top-left corner of the image.

```
var marker = new google.maps.Marker({
  position: new google.maps.LatLng(51.5, -0.2),
  icon: {
    url: 'https://azurermapscodesamples.azurewebsites.net/Common/images/icons/ylw-pushpin.png',
    anchor: new google.maps.Point(5, 30)
  },
  map: map
});
```



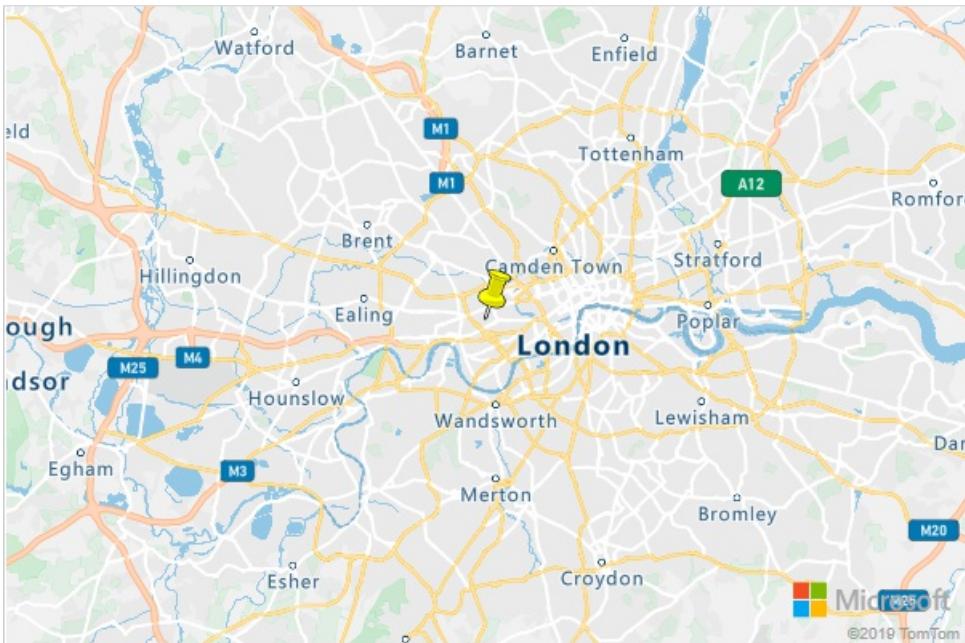
After: Azure Maps using HTML Markers

To customize an HTML marker, pass an HTML `string` or `HTMLElement` to the `htmlContent` option of the marker. Use the `anchor` option to specify the relative position of the marker, relative to the position coordinate. Assign one of nine defined reference points to the `anchor` option. Those defined points are: "center", "top", "bottom", "left", "right", "top-left", "top-right", "bottom-left", "bottom-right". The content is anchored to the bottom center of the html content by default. To make it easier to migrate code from Google Maps, set the `anchor` to "top-left", and then use the `pixelOffset` option with the same offset used in Google Maps. The offsets in Azure Maps move in the opposite direction of the offsets in Google Maps. So, multiply the offsets by minus one.

TIP

Add `pointer-events:none` as a style on the html content to disable the default drag behavior in Microsoft Edge, which will display an unwanted icon.

```
map.markers.add(new atlas.HtmlMarker({
    htmlContent: '',
    anchor: 'top-left',
    pixelOffset: [-5, -30],
    position: [-0.2, 51.5]
}));
```



After: Azure Maps using a Symbol Layer

Symbol layers in Azure Maps support custom images as well. First, load the image to the map resources and assign it with a unique ID. Reference the image in the symbol layer. Use the `offset` option to align the image to the correct point on the map. Use the `anchor` option to specify the relative position of the symbol, relative to the position coordinates. Use one of the nine defined reference points. Those points are: "center", "top", "bottom", "left", "right", "top-left", "top-right", "bottom-left", "bottom-right". The content is anchored to the bottom center of the html content by default. To make it easier to migrate code from Google Maps, set the `anchor` to "top-left", and then use the `offset` option with the same offset used in Google Maps. The offsets in Azure Maps move in the opposite direction of the offsets in Google Maps. So, multiply the offsets by minus one.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css"
type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <script type='text/javascript'>
        var map, datasource;

        function initMap() {
            map = new atlas.Map('myMap', {
                center: [-0.2, 51.5],
                zoom: 9,
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

            //Wait until the map resources are ready.
            map.events.add('ready', function () {

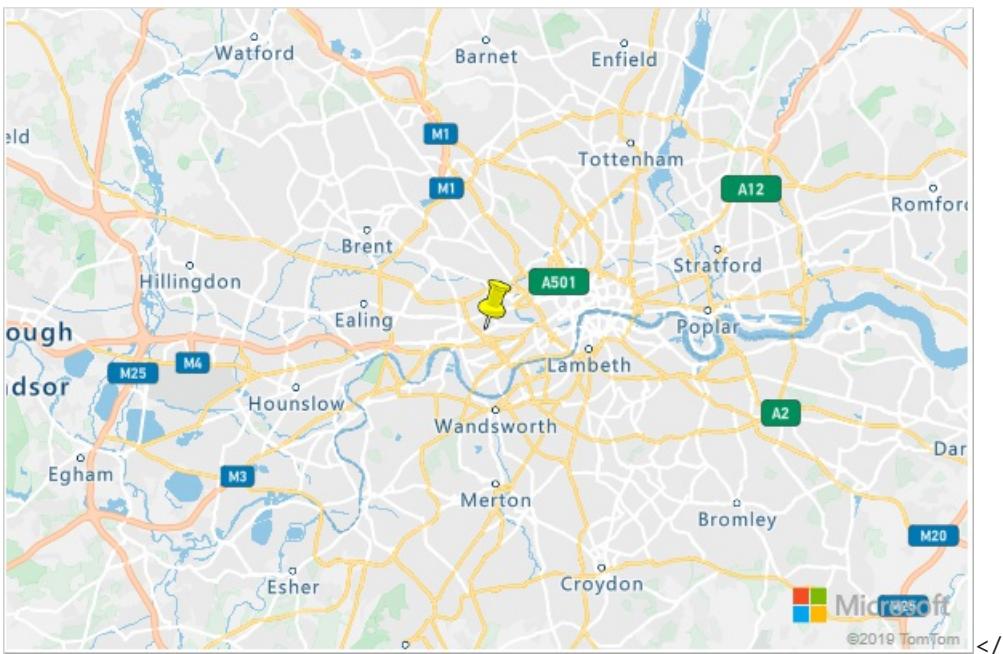
                //Load the custom image icon into the map resources.
                map.imageSprite.add('my-yellow-pin',
                    'https://azurermapscodeexamples.azurewebsites.net/Common/images/icons/ylw-pushpin.png').then(function () {

                        //Create a data source and add it to the map.
                        datasource = new atlas.source.DataSource();
                        map.sources.add(datasource);

                        //Create a point and add it to the data source.
                        datasource.add(new atlas.data.Point([-0.2, 51.5]));

                        //Add a layer for rendering point data as symbols.
                        map.layers.add(new atlas.layer.SymbolLayer(datasource, null, {
                            iconOptions: {
                                //Set the image option to the id of the custom icon that was loaded into the map
resources.
                                image: 'my-yellow-pin',
                                anchor: 'top-left',
                                offset: [-5, -30]
                            }
                        }));
                    });
                }
            });
        }
    </script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



TIP

To render advanced custom points, use multiple rendering layers together. For example, let's say you want to have multiple pushpins that have the same icon on different colored circles. Instead of creating a bunch of images for each color overlay, add a symbol layer on top of a bubble layer. Have the pushpins reference the same data source. This approach will be more efficient than creating and maintaining a bunch of different images.

Additional resources:

- [Create a data source](#)
- [Add a Symbol layer](#)
- [Add HTML Markers](#)
- [Use data-driven style expressions](#)
- [Symbol layer icon options](#)
- [Symbol layer text option](#)
- [HTML marker class](#)
- [HTML marker options](#)

Adding a polyline

Use polylines to represent a line or path on the map. Let's create a dashed polyline on the map.

Before: Google Maps

The Polyline class accepts a set of options. Pass an array of coordinates in the `path` option of the polyline.

```

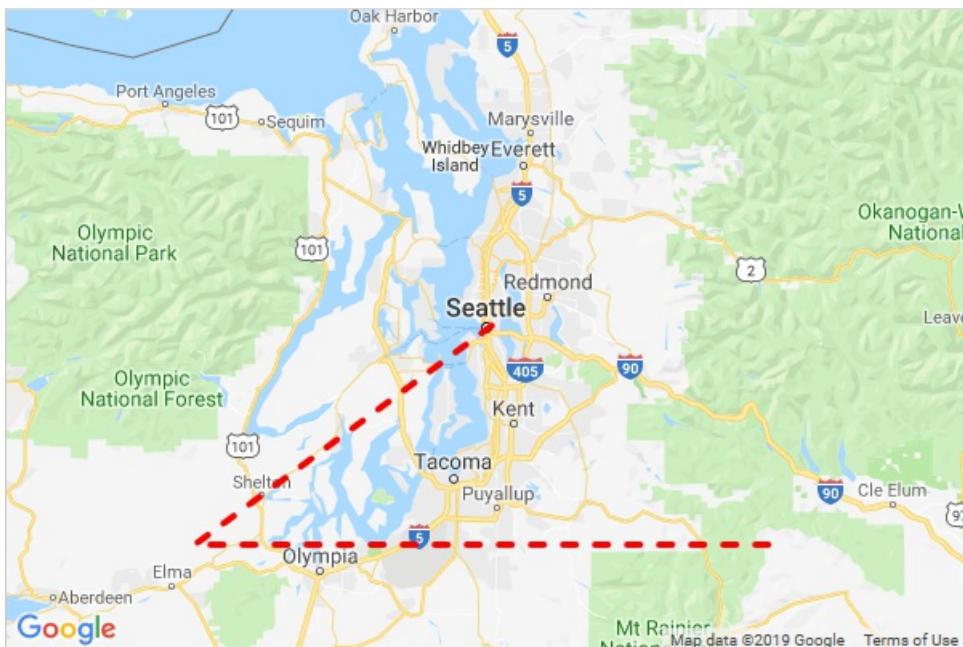
//Get the center of the map.
var center = map.getCenter();

//Define a symbol using SVG path notation, with an opacity of 1.
var lineSymbol = {
  path: 'M 0,-1 0,1',
  strokeOpacity: 1,
  scale: 4
};

//Create the polyline.
var line = new google.maps.Polyline({
  path: [
    center,
    new google.maps.LatLng(center.lat() - 0.5, center.lng() - 1),
    new google.maps.LatLng(center.lat() - 0.5, center.lng() + 1)
  ],
  strokeColor: 'red',
  strokeOpacity: 0,
  strokeWeight: 4,
  icons: [
    {
      icon: lineSymbol,
      offset: '0',
      repeat: '20px'
    }
  ]
});

//Add the polyline to the map.
line.setMap(map);

```



After: Azure Maps

Polylines are called `LineString` or `MultiLineString` objects. These objects can be added to a data source and rendered using a line layer. Add `LineString` to a data source, then add the data source to a `LineLayer` to render it.

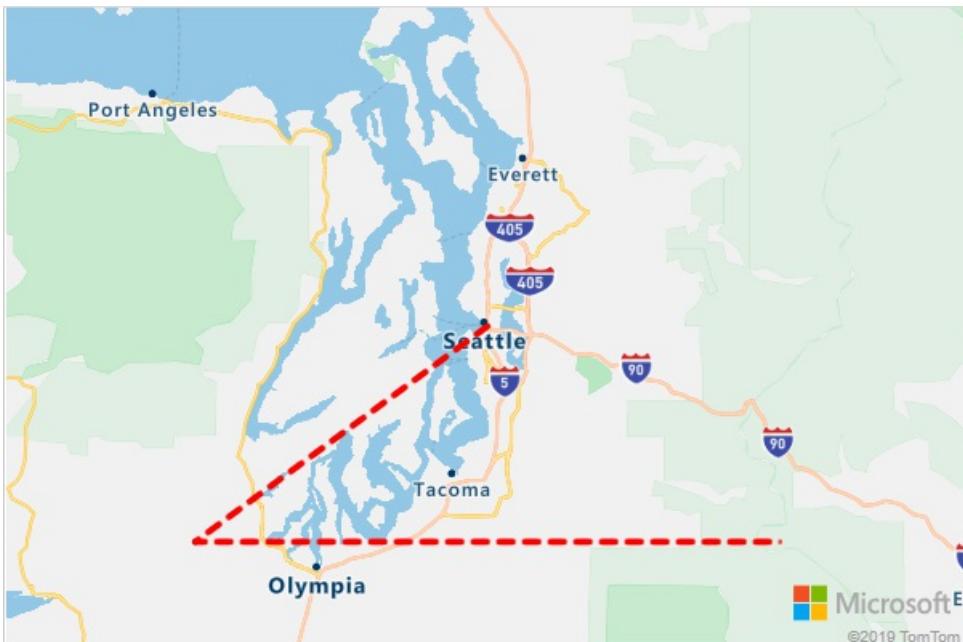
```

//Get the center of the map.
var center = map.getCamera().center;

//Create a data source and add it to the map.
var datasource = new atlas.source.DataSource();
map.sources.add(datasource);

//Create a line and add it to the data source.
datasource.add(new atlas.data.LineString([
    center,
    [center[0] - 1, center[1] - 0.5],
    [center[0] + 1, center[1] - 0.5]
]));

//Add a layer for rendering line data.
map.layers.add(new atlas.layer.LineLayer(datasource, null, {
    strokeColor: 'red',
    strokeWidth: 4,
    strokeDashArray: [3, 3]
}));
```



Additional resources:

- [Add lines to the map](#)
- [Line layer options](#)
- [Use data-driven style expressions](#)

Adding a polygon

Azure Maps and Google Maps provide similar support for polygons. Polygons are used to represent an area on the map. The following examples show how to create a polygon that forms a triangle based on the center coordinate of the map.

Before: Google Maps

The Polygon class accepts a set of options. Pass an array of coordinates to the `paths` option of the polygon.

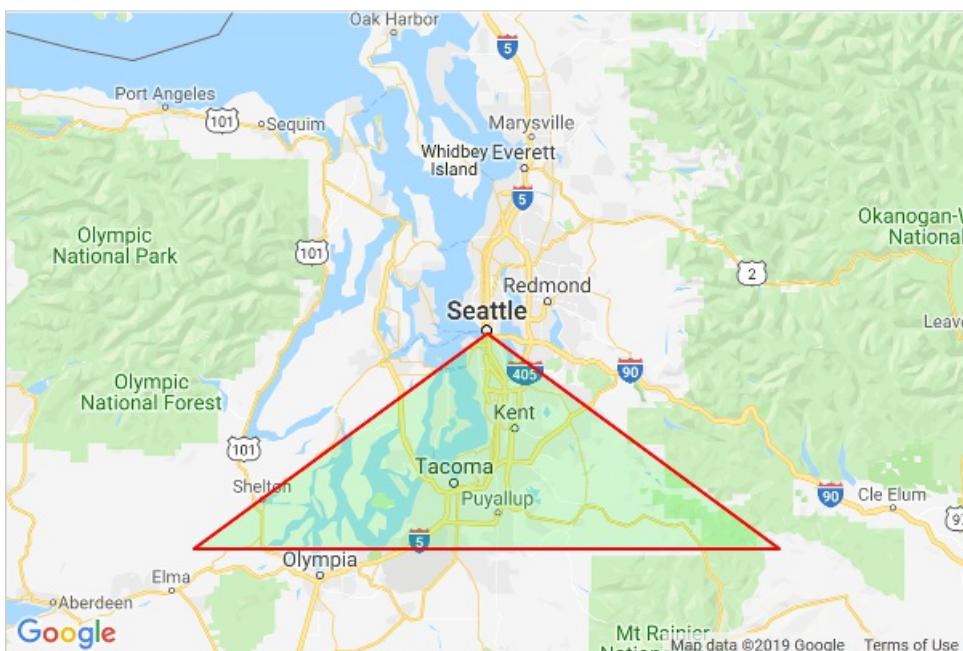
```

//Get the center of the map.
var center = map.getCenter();

//Create a polygon.
var polygon = new google.maps.Polygon({
  paths: [
    center,
    new google.maps.LatLng(center.lat() - 0.5, center.lng() - 1),
    new google.maps.LatLng(center.lat() - 0.5, center.lng() + 1),
    center
  ],
  strokeColor: 'red',
  strokeWeight: 2,
  fillColor: 'rgba(0, 255, 0, 0.5)'
});

//Add the polygon to the map
polygon.setMap(map);

```



After: Azure Maps

Add a `Polygon` or a `MultiPolygon` objects to a data source. Render the object on the map using layers. Render the area of a polygon using a polygon layer. And, render the outline of a polygon using a line layer.

```

//Get the center of the map.
var center = map.getCamera().center;

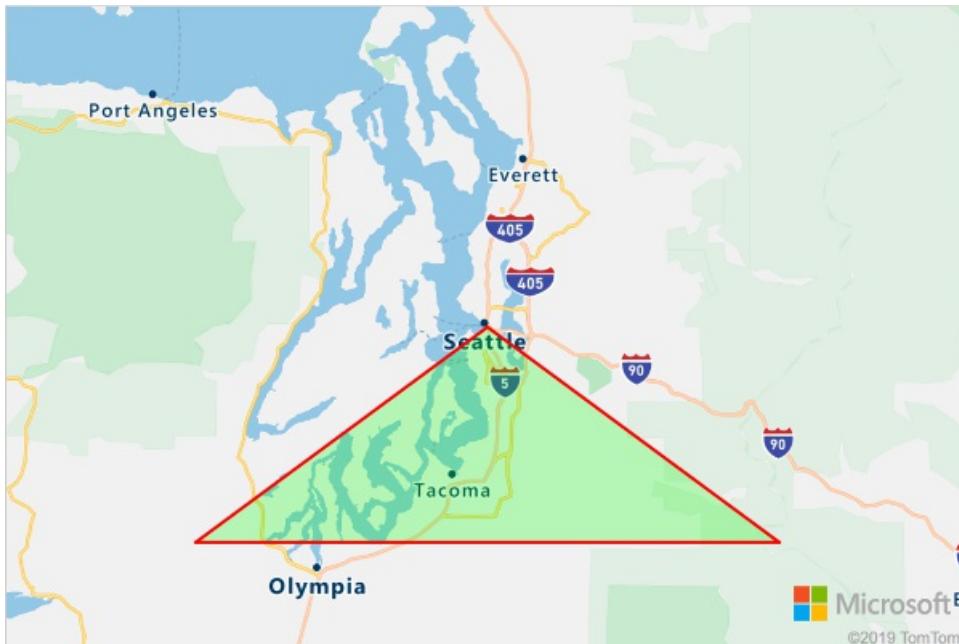
//Create a data source and add it to the map.
datasource = new atlas.source.DataSource();
map.sources.add(datasource);

//Create a polygon and add it to the data source.
datasource.add(new atlas.data.Polygon([
    center,
    [center[0] - 1, center[1] - 0.5],
    [center[0] + 1, center[1] - 0.5],
    center
]));

//Add a polygon layer for rendering the polygon area.
map.layers.add(new atlas.layer.PolygonLayer(datasource, null, {
    fillColor: 'rgba(0, 255, 0, 0.5)'
}));

//Add a line layer for rendering the polygon outline.
map.layers.add(new atlas.layer.LineLayer(datasource, null, {
    strokeColor: 'red',
    strokeWidth: 2
}));

```



Additional resources:

- [Add a polygon to the map](#)
- [Add a circle to the map](#)
- [Polygon layer options](#)
- [Line layer options](#)
- [Use data-driven style expressions](#)

Display an info window

Additional information for an entity can be displayed on the map as a `google.maps.InfoWindow` class in Google Maps. In Azure Maps, this functionality can be achieved using the `atlas.Popup` class. The next examples add a marker to the map. When the marker is clicked, an info window or a popup is displayed.

Before: Google Maps

Instantiate an info window using the `google.maps.InfoWindow` constructor.

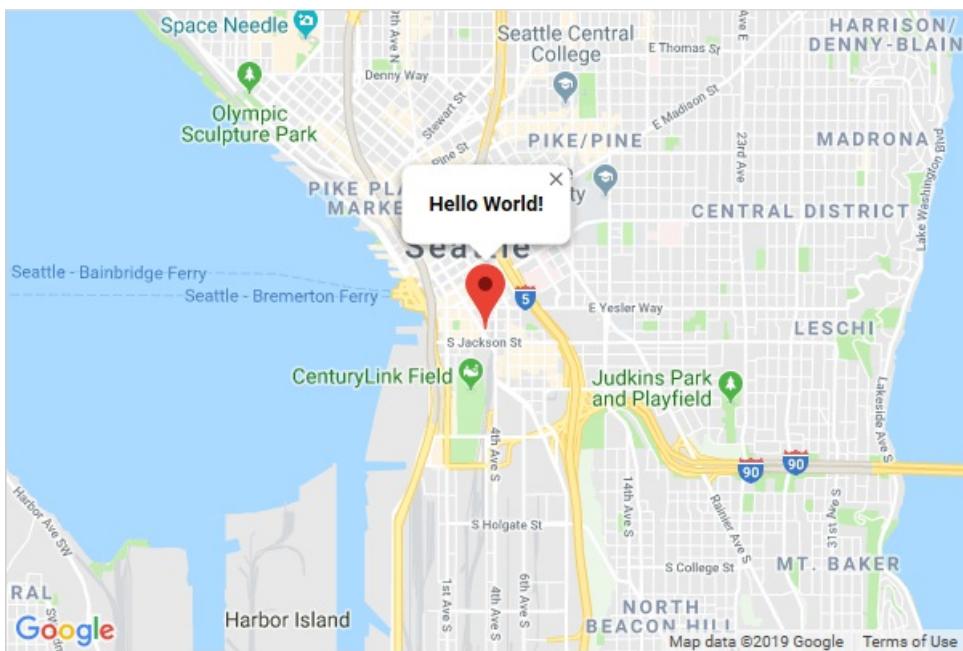
```

//Add a marker in which to display an infowindow for.
var marker = new google.maps.Marker({
  position: new google.maps.LatLng(47.6, -122.33),
  map: map
});

//Create an infowindow.
var infowindow = new google.maps.InfoWindow({
  content: '<div style="padding:5px"><b>Hello World!</b></div>'
});

//Add a click event to the marker to open the infowindow.
marker.addListener('click', function () {
  infowindow.open(map, marker);
});

```



After: Azure Maps

Let's use `popup` to display additional information about the location. Pass an `HTML string` or `HTMLElement` object to the `content` option of the `popup`. If you want, popups can be displayed independently of any shape. Thus, Popups require a `position` value to be specified. Specify the `position` value. To display a `popup`, call the `open` method and pass the `map` in which the `popup` is to be displayed on.

```

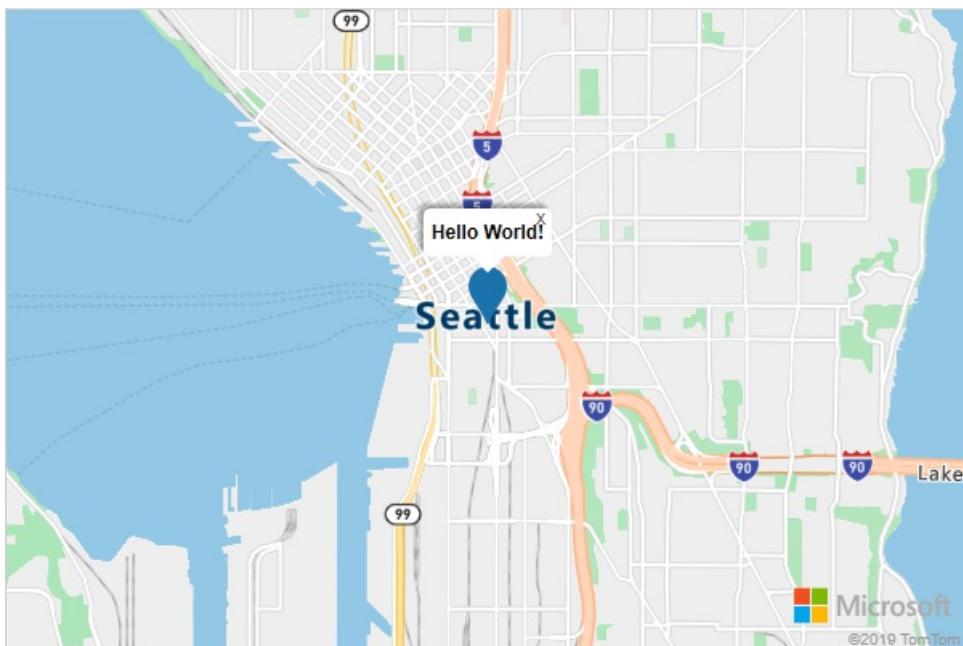
//Add a marker to the map in which to display a popup for.
var marker = new atlas.HtmlMarker({
    position: [-122.33, 47.6]
});

//Add the marker to the map.
map.markers.add(marker);

//Create a popup.
var popup = new atlas.Popup({
    content: '<div style="padding:5px"><b>Hello World!</b></div>',
    position: [-122.33, 47.6],
    pixelOffset: [0, -35]
});

//Add a click event to the marker to open the popup.
map.events.add('click', marker, function () {
    //Open the popup
    popup.open(map);
});

```



NOTE

You may do the same thing with a symbol, bubble, line or polygon layer by passing the chosen layer to the maps event code instead of a marker.

Additional resources:

- [Add a popup](#)
- [Popup with Media Content](#)
- [Popups on Shapes](#)
- [Reusing Popup with Multiple Pins](#)
- [Popup class](#)
- [Popup options](#)

Import a GeoJSON file

Google Maps supports loading and dynamically styling GeoJSON data via the `google.maps.Data` class. The functionality of this class aligns much more with the data-driven styling of Azure Maps. But, there's a key

difference. With Google Maps, you specify a callback function. The business logic for styling each feature it processed individually in the UI thread. But in Azure Maps, layers support specifying data-driven expressions as styling options. These expressions are processed at render time on a separate thread. The Azure Maps approach improves rendering performance. This advantage is noticed when larger data sets need to be rendered quickly.

The following examples load a GeoJSON feed of all earthquakes over the last seven days from the USGS. Earthquakes data renders as scaled circles on the map. The color and scale of each circle is based on the magnitude of each earthquake, which is stored in the `"mag"` property of each feature in the data set. If the magnitude is greater than or equal to five, the circle will be red. If it's greater or equal to three, but less than five, the circle will be orange. If it's less than three, the circle will be green. The radius of each circle will be the exponential of the magnitude multiplied by 0.1.

Before: Google Maps

Specify a single callback function in the `map.data.setStyle` method. Inside the callback function, apply business logic to each feature. Load the GeoJSON feed with the `map.data.loadGeoJson` method.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <script type='text/javascript'>
        var map;
        var earthquakeFeed = 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.geojson';

        function initMap() {
            map = new google.maps.Map(document.getElementById('myMap'), {
                center: new google.maps.LatLng(20, -160),
                zoom: 2
            });

            //Define a callback to style each feature.
            map.data.setStyle(function (feature) {

                //Extract the 'mag' property from the feature.
                var mag = parseFloat(feature.getProperty('mag'));

                //Set the color value to 'green' by default.
                var color = 'green';

                //If the mag value is greater than 5, set the color to 'red'.
                if (mag >= 5) {
                    color = 'red';
                }
                //If the mag value is greater than 3, set the color to 'orange'.
                else if (mag >= 3) {
                    color = 'orange';
                }

                return /** @type {google.maps.Data.StyleOptions} */({
                    icon: {
                        path: google.maps.SymbolPath.CIRCLE,

                        //Scale the radius based on an exponential of the 'mag' value.
                        scale: Math.exp(mag) * 0.1,
                        fillColor: color,
                        fillOpacity: 0.75,
                        strokeWeight: 2,
                        strokeColor: 'white'
                    }
                });
            });
        }

        //Load the data feed.
        map.data.loadGeoJson(earthquakeFeed);
    }
</script>

<!-- Google Maps Script Reference -->
<script src="https://maps.googleapis.com/maps/api/js?callback=initMap&key=[Your Google Maps Key]" async
defer></script>
</head>
<body>
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



After: Azure Maps

GeoJSON is the base data type in Azure Maps. Import it into a data source using the `datasource.importFromUrl` method. Use a bubble layer. The bubble layer provides functionality for rendering scaled circles, based on the properties of the features in a data source. Instead of having a callback function, the business logic is converted into an expression and passed into the style options. Expressions define how the business logic works. Expressions can be passed into another thread and evaluated against the feature data. Multiple data sources and layers can be added to Azure Maps, each with different business logic. This feature allows multiple data sets to be rendered on the map in different ways.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css"
type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <script type='text/javascript'>
        var map;
        var earthquakeFeed = 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.geojson';

        function initMap() {
            //Initialize a map instance.
            map = new atlas.Map('myMap', {
                center: [-160, 20],
                zoom: 1,
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

            //Wait until the map resources are ready.
            map.events.add('ready', function () {

```

```

//Create a data source and add it to the map.
datasource = new atlas.source.DataSource();
map.sources.add(datasource);

//Load the earthquake data.
datasource.importDataFromUrl(earthquakeFeed);

//Create a layer to render the data points as scaled circles.
map.layers.add(new atlas.layer.BubbleLayer(datasource, null, {
    //Make the circles semi-transparent.
    opacity: 0.75,

    color: [
        'case',
        //If the mag value is greater than 5, return 'red'.
        ['>=', ['get', 'mag'], 5],
        'red',
        //If the mag value is greater than 3, return 'orange'.
        ['>=', ['get', 'mag'], 3],
        'orange',
        //Return 'green' as a fallback.
        'green'
    ],
    //Scale the radius based on an exponential of the 'mag' value.
    radius: ['*', ['^', ['e'], ['get', 'mag']], 0.1]
}));

});

```

</script>

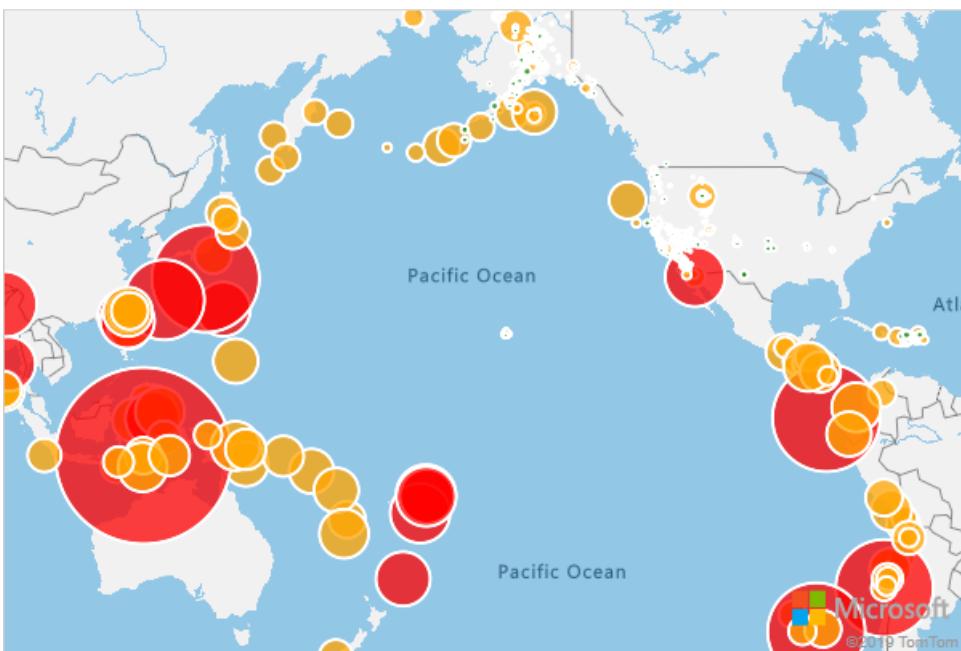
</head>

<body onload="initMap()">

<div id='myMap' style='position:relative;width:600px;height:400px;'></div>

</body>

</html>



Additional resources:

- [Add a Symbol layer](#)
- [Add a Bubble layer](#)
- [Cluster point data](#)

- [Use data-driven style expressions](#)

Marker clustering

When visualizing many data points on the map, points may overlap each other. Overlapping makes the map looks cluttered, and the map becomes difficult to read and use. Clustering point data is the process of combining data points that are near each other and representing them on the map as a single clustered data point. As the user zooms into the map, the clusters break apart into their individual data points. Cluster data points to improve user experience and map performance.

In the following examples, the code loads a GeoJSON feed of earthquake data from the past week and adds it to the map. Clusters are rendered as scaled and colored circles. The scale and color of the circles depends on the number of points they contain.

NOTE

Google Maps and Azure Maps use slightly different clustering algorithms. As such, sometimes the point distribution in the clusters varies.

Before: Google Maps

Use the MarkerCluster library to cluster markers. Cluster icons are limited to images, which have the numbers one through five as their name. They're hosted in the same directory.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <script type='text/javascript'>
        var map;
        var earthquakeFeed = 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.geojson';

        function initMap() {
            map = new google.maps.Map(document.getElementById('myMap'), {
                center: new google.maps.LatLng(20, -160),
                zoom: 2
            });

            //Download the GeoJSON data.
            fetch(earthquakeFeed)
                .then(function (response) {
                    return response.json();
                }).then(function (data) {
                    //Loop through the GeoJSON data and create a marker for each data point.
                    var markers = [];

                    for (var i = 0; i < data.features.length; i++) {

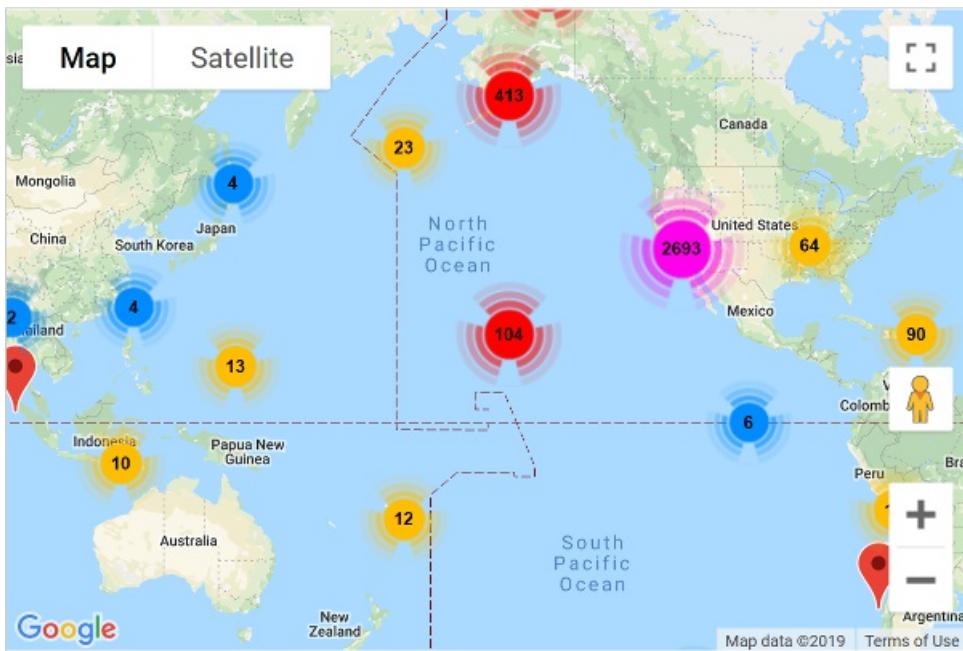
                        markers.push(new google.maps.Marker({
                            position: new google.maps.LatLng(data.features[i].geometry.coordinates[1],
data.features[i].geometry.coordinates[0])
                        }));
                    }

                    //Create a marker clusterer instance and tell it where to find the cluster icons.
                    var markerCluster = new MarkerClusterer(map, markers,
                    { imagePath:
'https://developers.google.com/maps/documentation/javascript/examples/markerclusterer/m' });
                });
            }
        </script>

        <!-- Load the marker cluster library. -->
        <script
src="https://developers.google.com/maps/documentation/javascript/examples/markerclusterer/markerclusterer.js"
></script>

        <!-- Google Maps Script Reference -->
        <script src="https://maps.googleapis.com/maps/api/js?callback=initMap&key=[Your Google Maps Key]" async
defer></script>
</head>
<body>
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



After: Azure Maps

Add and manage data in a data source. Connect data sources and layers, then render the data. The `DataSource` class in Azure Maps provides several clustering options.

- `cluster` – Tells the data source to cluster point data.
- `clusterRadius` - The radius in pixels to cluster points together.
- `clusterMaxZoom` - The maximum zoom level in which clustering occurs. If you zoom in more than this level, all points are rendered as symbols.
- `clusterProperties` - Defines custom properties that are calculated using expressions against all the points within each cluster and added to the properties of each cluster point.

When clustering is enabled, the data source will send clustered and unclustered data points to layers for rendering. The data source is capable of clustering hundreds of thousands of data points. A clustered data point has the following properties:

PROPERTY NAME	TYPE	DESCRIPTION
<code>cluster</code>	boolean	Indicates if feature represents a cluster.
<code>cluster_id</code>	string	A unique ID for the cluster that can be used with the <code>DataSource</code> <code>getClusterExpansionZoom</code> , <code>getClusterChildren</code> , and <code>getClusterLeaves</code> methods.
<code>point_count</code>	number	The number of points the cluster contains.
<code>point_count_abbreviated</code>	string	A string that abbreviates the <code>point_count</code> value if it is long. (for example, 4,000 becomes 4K)

The `DataSource` class has the following helper function for accessing additional information about a cluster using the `cluster_id`.

METHOD	RETURN TYPE	DESCRIPTION
getClusterChildren(clusterId: number)	Promise<Array<Feature<Geometry, any> Shape>>	Retrieves the children of the given cluster on the next zoom level. These children may be a combination of shapes and subclusters. The subclusters will be features with properties matching ClusteredProperties.
getClusterExpansionZoom(clusterId: number)	Promise<number>	Calculates a zoom level at which the cluster will start expanding or break apart.
getClusterLeaves(clusterId: number, limit: number, offset: number)	Promise<Array<Feature<Geometry, any> Shape>>	Retrieves all points in a cluster. Set the <code>limit</code> to return a subset of the points, and use the <code>offset</code> to page through the points.

When rendering clustered data on the map, it's often best to use two or more layers. The following example uses three layers. A bubble layer for drawing scaled colored circles based on the size of the clusters. A symbol layer to render the cluster size as text. And, it uses a second symbol layer for rendering the unclustered points. There are many other ways to render clustered data. For more information, see the [Cluster point data](#) documentation.

Directly import GeoJSON data using the `importDataFromUrl` function on the `DataSource` class, inside Azure Maps map.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <script type='text/javascript'>
        var map, datasource;
        var earthquakeFeed = 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.geojson';

        function initMap() {
            //Initialize a map instance.
            map = new atlas.Map('myMap', {
                center: [-160, 20],
                zoom: 1,
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

            //Wait until the map resources are ready.
            map.events.add('ready', function () {

                //Create a data source and add it to the map.
                datasource = new atlas.source.DataSource(null, {

```

```

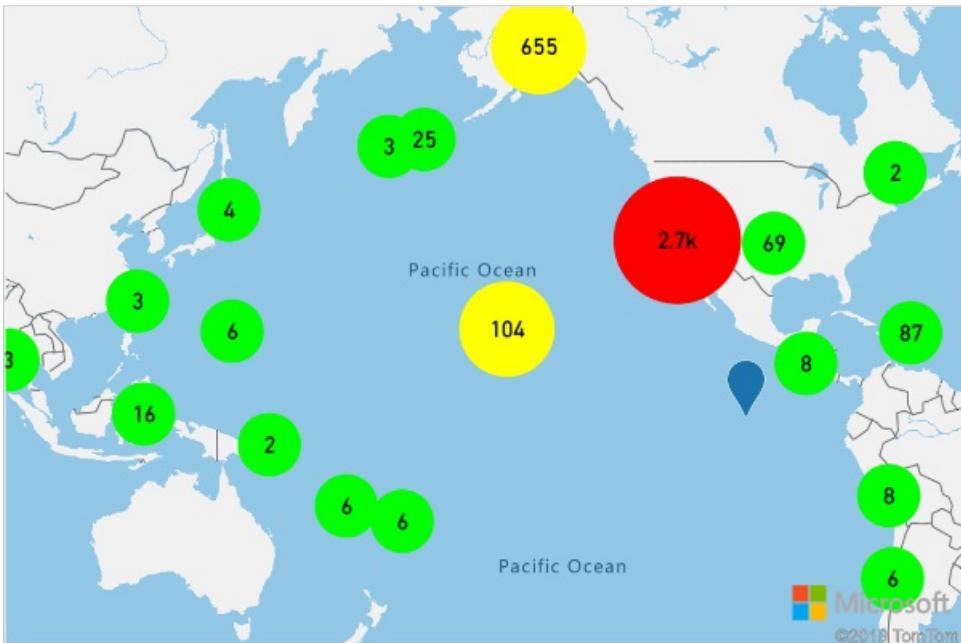
        //Tell the data source to cluster point data.
        cluster: true
    });
    map.sources.add(datasource);

    //Create layers for rendering clusters, their counts and unclustered points and add the
    layers to the map.
    map.layers.add([
        //Create a bubble layer for rendering clustered data points.
        new atlas.layer.BubbleLayer(datasource, null, {
            //Scale the size of the clustered bubble based on the number of points inthe
            cluster.
            radius: [
                'step',
                ['get', 'point_count'],
                20,           //Default of 20 pixel radius.
                100, 30,     //If point_count >= 100, radius is 30 pixels.
                750, 40     //If point_count >= 750, radius is 40 pixels.
            ],
            //Change the color of the cluster based on the value on the point_cluster property
            of the cluster.
            color: [
                'step',
                ['get', 'point_count'],
                'lime',          //Default to lime green.
                100, 'yellow',   //If the point_count >= 100, color is yellow.
                750, 'red'       //If the point_count >= 750, color is red.
            ],
            strokeWidth: 0,
            filter: ['has', 'point_count'] //Only rendered data points which have a point_count
            property, which clusters do.
        }),

        //Create a symbol layer to render the count of locations in a cluster.
        new atlas.layer.SymbolLayer(datasource, null, {
            iconOptions: {
                image: 'none' //Hide the icon image.
            },
            textOptions: {
                textField: ['get', 'point_count_abbreviated'],
                offset: [0, 0.4]
            }
        }),
        //Create a layer to render the individual locations.
        new atlas.layer.SymbolLayer(datasource, null, {
            filter: ['!', ['has', 'point_count']] //Filter out clustered points from this layer.
        })
    ]);

    //Retrieve a GeoJSON data set and add it to the data source.
    datasource.importDataFromUrl(earthquakeFeed);
});
}
</script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



Additional resources:

- [Add a Symbol layer](#)
- [Add a Bubble layer](#)
- [Cluster point data](#)
- [Use data-driven style expressions](#)

Add a heat map

Heat maps, also known as point density maps, are a type of data visualization. They're used to represent the density of data using a range of colors. And, they're often used to show the data "hot spots" on a map. Heat maps are a great way to render large point data sets.

The following examples load a GeoJSON feed of all earthquakes over the past month, from the USGS, and it renders them as a weighted heat map. The `"mag"` property is used as the weight.

Before: Google Maps

To create a heat map, load the "visualization" library by adding `&libraries=visualization` to the API script URL. The heat map layer in Google Maps doesn't support GeoJSON data directly. First, download the data and convert it into an array of weighted data points:

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <script type='text/javascript'>
        var map;
        var earthquakeFeed = 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_month.geojson';

        function initMap() {

            var map = new google.maps.Map(document.getElementById('myMap'), {
                center: new google.maps.LatLng(20, -160),
                zoom: 2,
                mapTypeId: 'satellite'
            });

            //Download the GeoJSON data.
            fetch(url).then(function (response) {
                return response.json();
            }).then(function (res) {
                var points = getDataPoints(res);

                var heatmap = new google.maps.visualization.HeatmapLayer({
                    data: points
                });
                heatmap.setMap(map);
            });
        }

        function getDataPoints(geojson, weightProp) {
            var points = [];

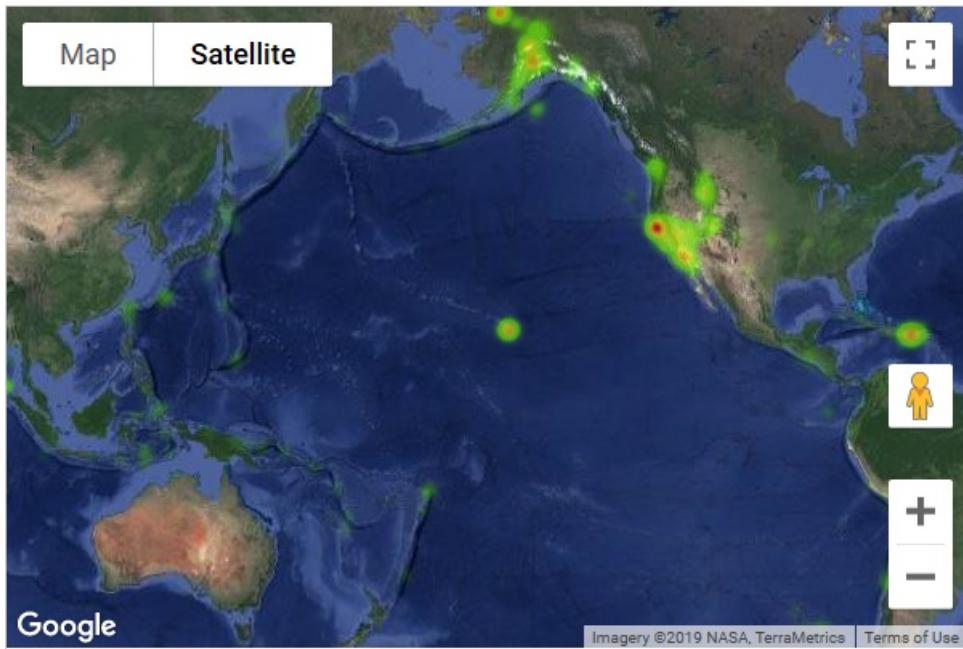
            for (var i = 0; i < geojson.features.length; i++) {
                var f = geojson.features[i];

                if (f.geometry.type === 'Point') {
                    points.push({
                        location: new google.maps.LatLng(f.geometry.coordinates[1],
f.geometry.coordinates[0]),
                        weight: f.properties[weightProp]
                    });
                }
            }

            return points;
        }
    </script>

    <!-- Google Maps Script Reference -->
    <script src="https://maps.googleapis.com/maps/api/js?callback=initMap&key=[Your Google Maps
Key]&libraries=visualization" async defer></script>
</head>
<body>
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



After: Azure Maps

Load the GeoJSON data into a data source and connect the data source to a heat map layer. The property that will be used for the weight can be passed into the `weight` option using an expression. Directly import GeoJSON data to Azure Maps using the `importDataFromUrl` function on the `DataSource` class.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <script type='text/javascript'>
        var map;
        var earthquakeFeed = 'https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_month.geojson';

        function initMap() {
            //Initialize a map instance.
            map = new atlas.Map('myMap', {
                center: [-160, 20],
                zoom: 1,
                style: 'satellite',
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

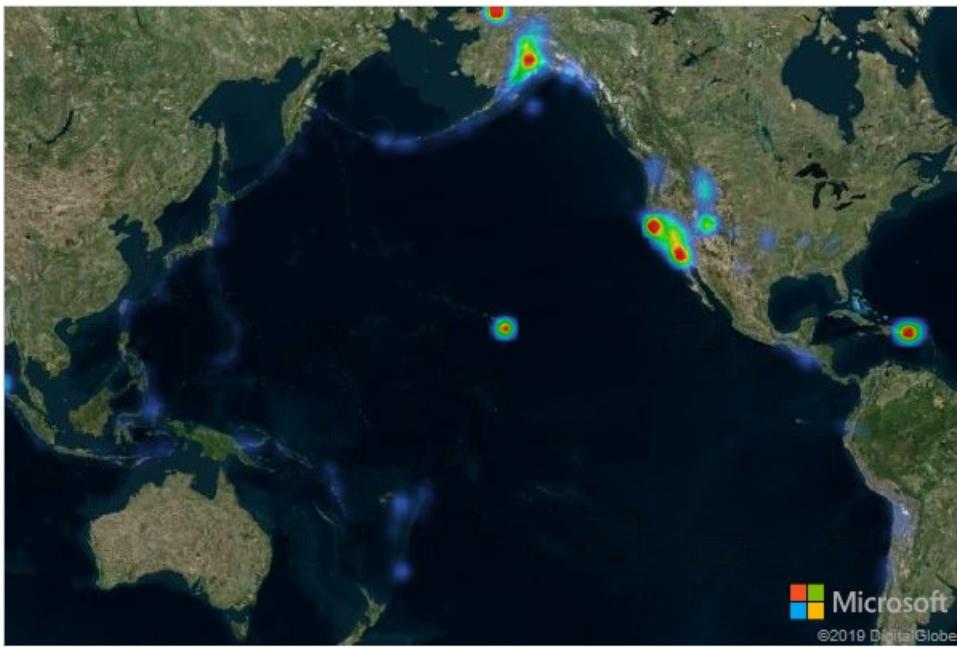
            //Wait until the map resources are ready.
            map.events.add('ready', function () {

                //Create a data source and add it to the map.
                datasource = new atlas.source.DataSource();
                map.sources.add(datasource);

                //Load the earthquake data.
                datasource.importDataFromUrl(earthquakeFeed);

                //Create a layer to render the data points as a heat map.
                map.layers.add(new atlas.layer.HeatMapLayer(datasource, null, {
                    weight: ['get', 'mag'],
                    intensity: 0.005,
                    opacity: 0.65,
                    radius: 10
                }));
            });
        }
    </script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



Additional resources:

- [Add a heat map layer](#)
- [Heat map layer class](#)
- [Heat map layer options](#)
- [Use data-driven style expressions](#)

Overlay a tile layer

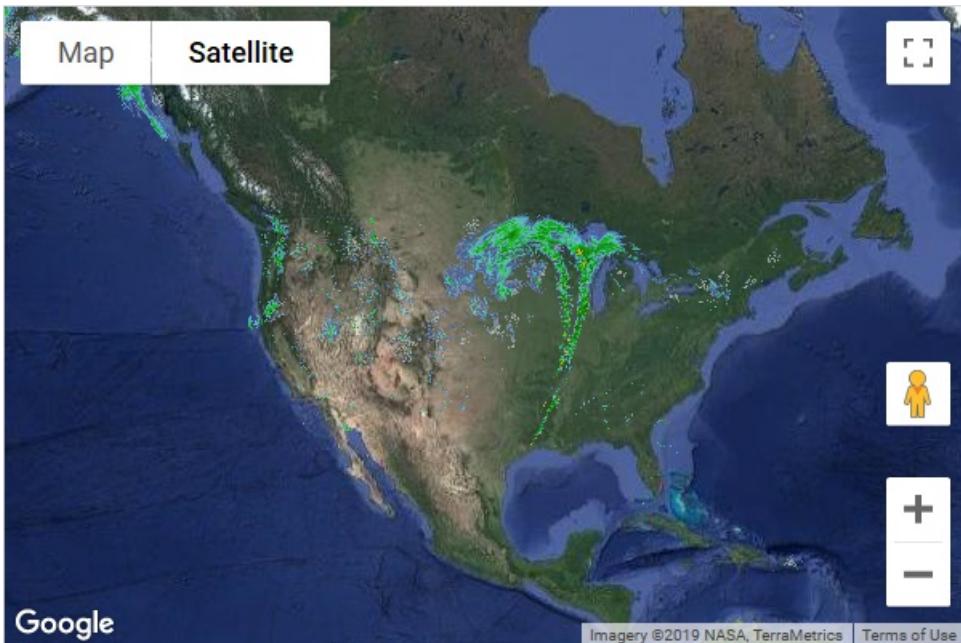
Tile layers in Azure Maps are known as Image overlays in Google Maps. Tile layers allow you to overlay large images that have been broken up into smaller tiled images, which align with the maps tiling system. This approach is commonly used to overlay large images or large data sets.

The following examples overlay a weather radar tile layer from Iowa Environmental Mesonet of Iowa State University.

Before: Google Maps

In Google Maps, tile layers can be created by using the `google.maps.ImageMapType` class.

```
map.overlayMapTypes.insertAt(0, new google.maps.ImageMapType({
    getTileUrl: function (coord, zoom) {
        return "https://mesonet.agron.iastate.edu/cache/tile.py/1.0.0/nexrad-n0q-900913/" + zoom + "/" +
        coord.x + "/" + coord.y;
    },
    tileSize: new google.maps.Size(256, 256),
    opacity: 0.8
}));
```



After: Azure Maps

Add a tile layer to the map similarly as any other layer. Use a formatted URL that has in x, y, zoom placeholders; `{x}` , `{y}` , `{z}` to tell the layer where to access the tiles. Azure Maps tile layers also support `{quadkey}` , `{bbox-epsg-3857}` , and `{subdomain}` placeholders.

TIP

In Azure Maps layers can easily be rendered below other layers, including base map layers. Often it is desirable to render tile layers below the map labels so that they are easy to read. The `map.layers.add` method takes in a second parameter which is the id of the layer in which to insert the new layer below. To insert a tile layer below the map labels, use this code:

```
map.layers.add(myTileLayer, "labels");
```

```
//Create a tile layer and add it to the map below the label layer.  
map.layers.add(new atlas.layer.TileLayer({  
    tileUrl: 'https://mesonet.agron.iastate.edu/cache/tile.py/1.0.0/nexrad-n0q-900913/{z}/{x}/{y}.png',  
    opacity: 0.8,  
    tileSize: 256  
}), 'labels');
```



TIP

Tile requests can be captured using the `transformRequest` option of the map. This will allow you to modify or add headers to the request if desired.

Additional resources:

- [Add tile layers](#)
- [Tile layer class](#)
- [Tile layer options](#)

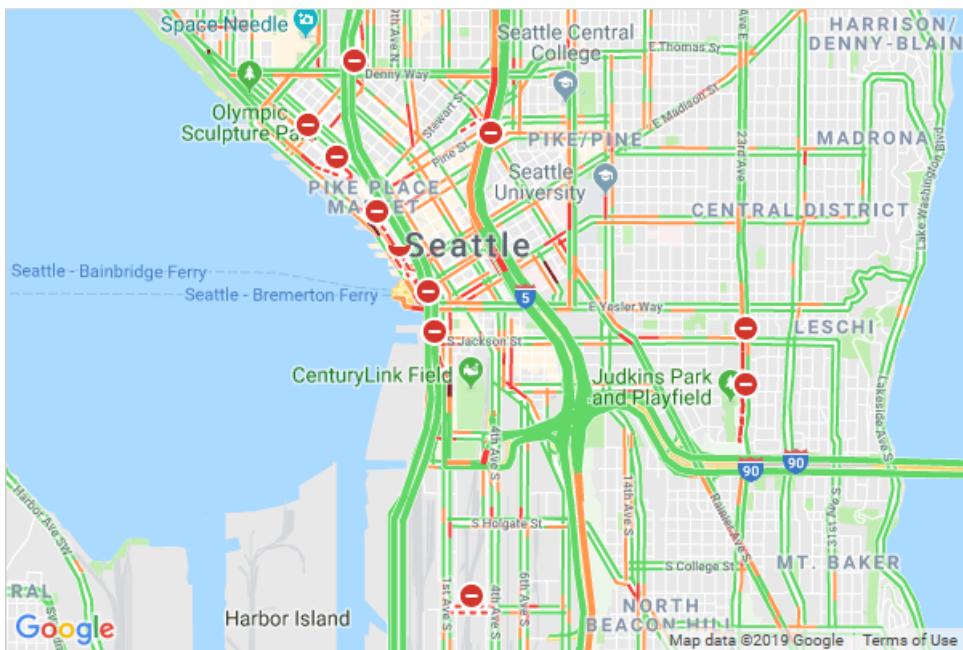
Show traffic data

Traffic data can be overlaid both Azure and Google maps.

Before: Google Maps

Overlay traffic data on the map using the traffic layer.

```
var trafficLayer = new google.maps.TrafficLayer();
trafficLayer.setMap(map);
```

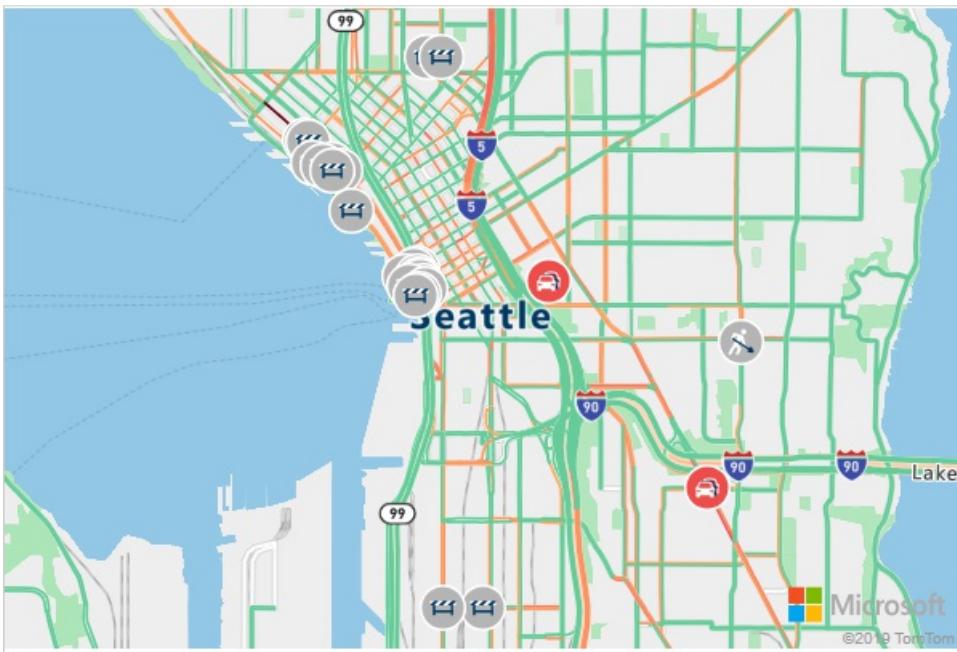


After: Azure Maps

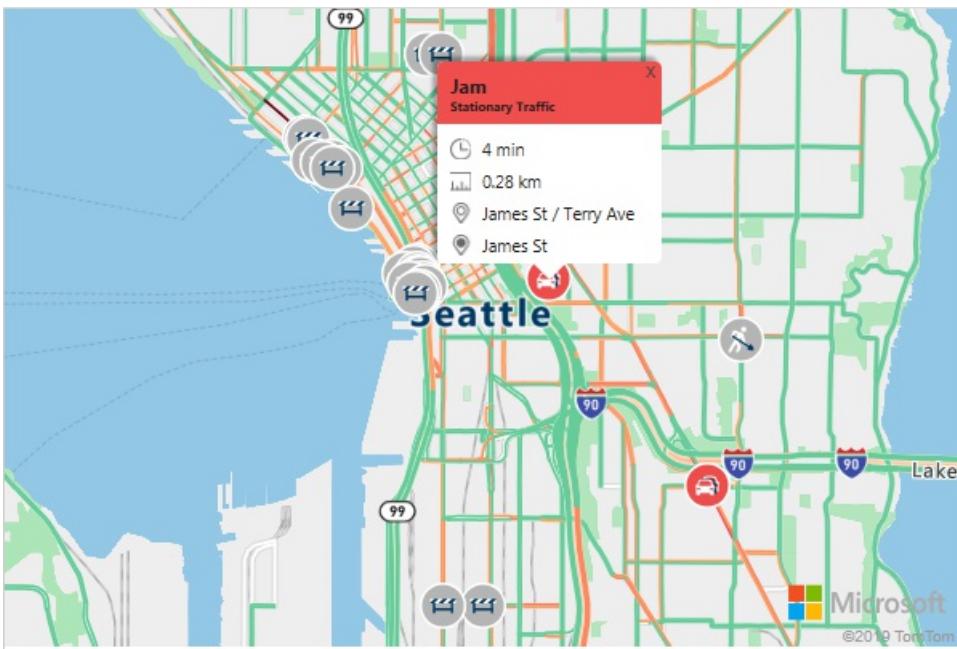
Azure Maps provides several different options for displaying traffic. Display traffic incidents, such as road closures and accidents as icons on the map. Overlay traffic flow and color coded roads on the map. The colors can be modified based on the posted speed limit, relative to the normal expected delay, or absolute delay. Incident data in Azure Maps updates every minute, and flow data updates every two minutes.

Assign the wanted values for `setTraffic` options.

```
map.setTraffic({
    incidents: true,
    flow: 'relative'
});
```



If you click on one of the traffic icons in Azure Maps, additional information is displayed in a popup.



Additional resources:

- [Show traffic on the map](#)
- [Traffic overlay options](#)

Add a ground overlay

Both Azure and Google maps support overlaying georeferenced images on the map. Georeferenced images move and scale as you pan and zoom the map. In Google Maps, georeferenced images are known as ground overlays while in Azure Maps they're referred to as image layers. They are great for building floor plans, overlaying old maps, or imagery from a drone.

Before: Google Maps

Specify the URL to the image you want to overlay and a bounding box to bind the image on the map. This example overlays a map image of [Newark New Jersey from 1922](#) on the map.

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <script type='text/javascript'>
        var map, historicalOverlay;

        function initMap() {
            map = new google.maps.Map(document.getElementById('myMap'), {
                center: new google.maps.LatLng(40.740, -74.18),
                zoom: 12
            });

            var imageBounds = {
                north: 40.773941,
                south: 40.712216,
                east: -74.12544,
                west: -74.22655
            };

            historicalOverlay = new google.maps.GroundOverlay(
                'https://www.lib.utexas.edu/maps/historical/newark_nj_1922.jpg',
                imageBounds);
            historicalOverlay.setMap(map);
        }
    </script>

    <!-- Google Maps Script Reference -->
    <script src="https://maps.googleapis.com/maps/api/js?callback=initMap&key=[Your Google Maps Key]" async defer></script>
</head>
<body>
    <div id="myMap" style="position:relative;width:600px;height:400px;"></div>
</body>
</html>

```

Running this code in a browser will display a map that looks like the following image:



After: Azure Maps

Use the `atlas.layer.ImageLayer` class to overlay georeferenced images. This class requires a URL to an image

and a set of coordinates for the four corners of the image. The image must be hosted either on the same domain or have CORs enabled.

TIP

If you only have north, south, east, west and rotation information, and you do not have coordinates for each corner of the image, you can use the static `atlas.layer.ImageLayer.getCoordinatesFromEdges` method.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <script type='text/javascript'>
        var map;

        function initMap() {
            //Initialize a map instance.
            map = new atlas.Map('myMap', {
                center: [-74.18, 40.740],
                zoom: 12,

                //Add your Azure Maps subscription key to the map SDK. Get an Azure Maps key at
                https://azure.com/maps
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

            //Wait until the map resources are ready.
            map.events.add('ready', function () {

                //Create an image layer and add it to the map.
                map.layers.add(new atlas.layer.ImageLayer({
                    url: 'newark_nj_1922.jpg',
                    coordinates: [
                        [-74.22655, 40.773941], //Top Left Corner
                        [-74.12544, 40.773941], //Top Right Corner
                        [-74.12544, 40.712216], //Bottom Right Corner
                        [-74.22655, 40.712216] //Bottom Left Corner
                    ]
                }));
            });
        }
    </script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>
```



Additional resources:

- [Overlay an image](#)
- [Image layer class](#)

Add KML data to the map

Both Azure and Google maps can import and render KML, KMZ and GeoRSS data on the map. Azure Maps also supports GPX, GML, spatial CSV files, GeoJSON, Well Known Text (WKT), Web-Mapping Services (WMS), Web-Mapping Tile Services (WMPS), and Web Feature Services (WFS). Azure Maps reads the files locally into memory and in most cases can handle much larger KML files.

Before: Google Maps

```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <script type='text/javascript'>
        var map, historicalOverlay;

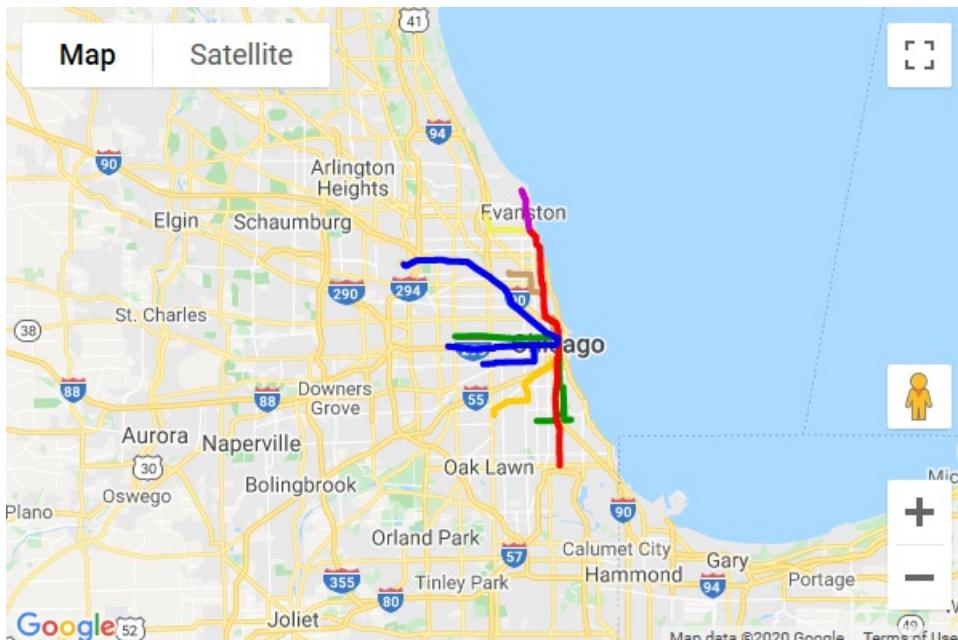
        function initMap() {
            map = new google.maps.Map(document.getElementById('myMap'), {
                center: new google.maps.LatLng(0, 0),
                zoom: 1
            });

            var layer = new google.maps.KmlLayer({
                url: 'https://googlearchive.github.io/js-v2-samples/ggeoxml/cta.kml',
                map: map
            });
        }
    </script>

    <!-- Google Maps Script Reference -->
    <script src="https://maps.googleapis.com/maps/api/js?callback=initMap&key=[Your Google Maps Key]" async defer></script>
</head>
<body>
    <div id="myMap" style="position:relative;width:600px;height:400px;"></div>
</body>
</html>

```

Running this code in a browser will display a map that looks like the following image:



After: Azure Maps

In Azure Maps, GeoJSON is the main data format used in the web SDK, additional spatial data formats can be easily integrated in using the [spatial IO module](#). This module has functions for both reading and writing spatial data and also includes a simple data layer which can easily render data from any of these spatial data formats. To read the data in a spatial data file, pass in a URL, or raw data as string or blob into the `atlas.io.read` function. This will return all the parsed data from the file that can then be added to the map. KML is a bit more complex than most spatial data format as it includes a lot more styling information. The `SpatialDataLayer` class

supports rendering majority of these styles, however icons/images have to be loaded into the map before loading the feature data, and ground overlays have to be added as layers to the map separately. When loading data via a URL, it should be hosted on a CORS enabled endpoint, or a proxy service should be passed in as an option into the read function.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=Edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

    <!-- Add reference to the Azure Maps Spatial IO module. -->
    <script src="https://atlas.microsoft.com/sdk/javascript/spatial/0/atlas-spatial.js"></script>

<script type='text/javascript'>
    var map, datasource, layer;

    function initMap() {
        //Initialize a map instance.
        map = new atlas.Map('myMap', {
            view: 'Auto',

            //Add your Azure Maps subscription key to the map SDK. Get an Azure Maps key at
            https://azure.com/maps
            authOptions: {
                authType: 'subscriptionKey',
                subscriptionKey: '<Your Azure Maps Key>'
            }
        });

        //Wait until the map resources are ready.
        map.events.add('ready', function () {

            //Create a data source and add it to the map.
            datasource = new atlas.source.DataSource();
            map.sources.add(datasource);

            //Add a simple data layer for rendering the data.
            layer = new atlas.layer.SimpleDataLayer(datasource);
            map.layers.add(layer);

            //Read a KML file from a URL or pass in a raw KML string.
            atlas.io.read('https://googlearchive.github.io/js-v2-samples/ggeoxml/cta.kml').then(async r => {
                if (r) {

                    //Check to see if there are any icons in the data set that need to be loaded into
                    //the map resources.
                    if (r.icons) {
                        //For each icon image, create a promise to add it to the map, then run the
                        promises in parallel.
                        var imagePromises = [];

                        //The keys are the names of each icon image.
                        var keys = Object.keys(r.icons);

                        if (keys.length !== 0) {
                            keys.forEach(function (key) {
                                imagePromises.push(map.imageSprite.add(key, r.icons[key]));
                            });
                        }
                    }
                }
            });
        });
    }
</script>
```

```

        await Promise.all(imagePromises);
    }
}

//Load all features.
if (r.features && r.features.length > 0) {
    datasource.add(r.features);
}

//Load all ground overlays.
if (r.groundOverlays && r.groundOverlays.length > 0) {
    map.layers.add(r.groundOverlays);
}

//If bounding box information is known for data, set the map view to it.
if (r.bbox) {
    map.setCamera({ bounds: r.bbox, padding: 50 });
}
}

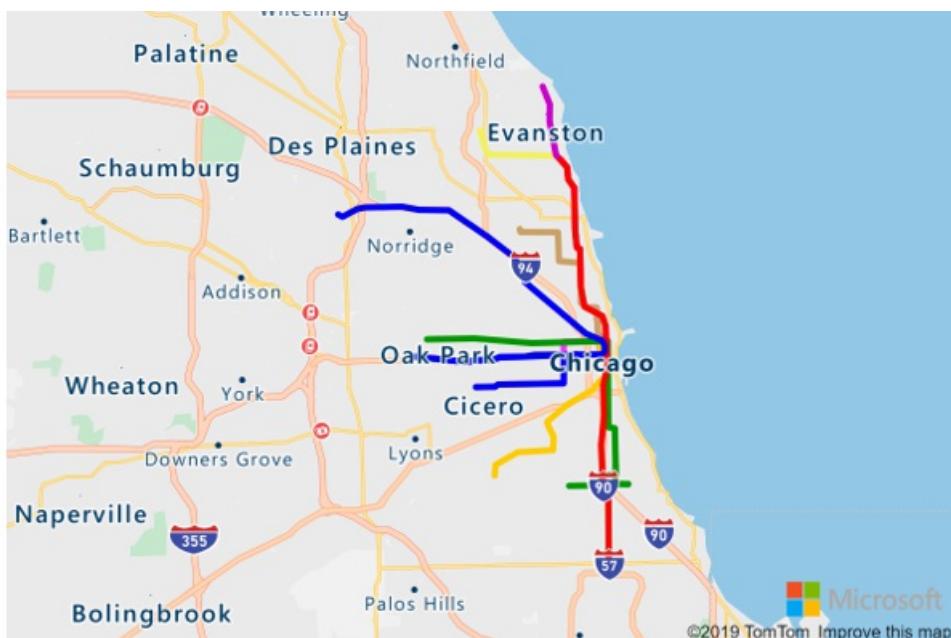
});

});

}

</script>
</head>
<body onload="initMap()">
    <div id='myMap' style='position:relative;width:600px;height:400px;'></div>
</body>
</html>

```



Additional resources:

- [atlas.io.read function](#)
- [SimpleDataLayer](#)
- [SimpleDataLayerOptions](#)

Additional code samples

The following are some additional code samples related to Google Maps migration:

- [Drawing tools](#)
- [Limit Map to Two Finger Panning](#)

- Limit Scroll Wheel Zoom
- Create a Fullscreen Control

Services:

- Using the Azure Maps services module
- Search for points of interest
- Get information from a coordinate (reverse geocode)
- Show directions from A to B
- Search Autosuggest with JQuery UI

Google Maps V3 to Azure Maps Web SDK class mapping

The following appendix provides a cross reference of the commonly used classes in Google Maps V3 and the equivalent Azure Maps Web SDK.

Core Classes

GOOGLE MAPS	AZURE MAPS
<code>google.maps.Map</code>	<code>atlas.Map</code>
<code>google.maps.InfoWindow</code>	<code>atlas.Popup</code>
<code>google.maps.InfoWindowOptions</code>	<code>atlas.PopupOptions</code>
<code>google.maps.LatLng</code>	<code>atlas.data.Position</code>
<code>google.maps.LatLngBounds</code>	<code>atlas.data.BoundingBox</code>
<code>google.maps.MapOptions</code>	<code>atlas.CameraOptions</code> <code>atlas.CameraBoundsOptions</code> <code>atlas.ServiceOptions</code> <code>atlas.StyleOptions</code> <code>atlas.UserInteractionOptions</code>
<code>google.maps.Point</code>	<code>atlas.Pixel</code>

Overlay Classes

GOOGLE MAPS	AZURE MAPS
<code>google.maps.Marker</code>	<code>atlas.HtmlMarker</code> <code>atlas.data.Point</code>
<code>google.maps.MarkerOptions</code>	<code>atlas.HtmlMarkerOptions</code> <code>atlas.layer.SymbolLayer</code> <code>atlas.SymbolLayerOptions</code> <code>atlas.IconOptions</code> <code>atlas.TextOptions</code> <code>atlas.layer.BubbleLayer</code> <code>atlas.BubbleLayerOptions</code>
<code>google.maps.Polygon</code>	<code>atlas.data.Polygon</code>

GOOGLE MAPS	AZURE MAPS
<code>google.maps.PolygonOptions</code>	<code>atlas.layer.PolygonLayer</code> <code>atlas.PolygonLayerOptions</code> <code>atlas.layer.LineLayer</code> <code>atlas.LineLayerOptions</code>
<code>google.maps.Polyline</code>	<code>atlas.data.LineString</code>
<code>google.maps.PolylineOptions</code>	<code>atlas.layer.LineLayer</code> <code>atlas.LineLayerOptions</code>
<code>google.maps.Circle</code>	See Add a circle to the map
<code>google.maps.ImageMapType</code>	<code>atlas.TileLayer</code>
<code>google.maps.ImageMapTypeOptions</code>	<code>atlas.TileLayerOptions</code>
<code>google.maps.GroundOverlay</code>	<code>atlas.layer.ImageLayer</code> <code>atlas.ImageLayerOptions</code>

Service Classes

The Azure Maps Web SDK includes a services module, which can be loaded separately. This module wraps the Azure Maps REST services with a web API and can be used in JavaScript, TypeScript, and Node.js applications.

GOOGLE MAPS	AZURE MAPS
<code>google.maps.Geocoder</code>	<code>atlas.service.SearchUrl</code>
<code>google.maps.GeocoderRequest</code>	<code>atlas.SearchAddressOptions</code> <code>atlas.SearchAddressReverseOptions</code> <code>atlas.SearchAddressReverseCrossStreetOptions</code> <code>atlas.SearchAddressStructuredOptions</code> <code>atlas.SearchAlongRouteOptions</code> <code>atlas.SearchFuzzyOptions</code> <code>atlas.SearchInsideGeometryOptions</code> <code>atlas.SearchNearbyOptions</code> <code>atlas.SearchPOIOptions</code> <code>atlas.SearchPOICategoryOptions</code>
<code>google.maps.DirectionsService</code>	<code>atlas.service.RouteUrl</code>
<code>google.maps.DirectionsRequest</code>	<code>atlas.CalculateRouteDirectionsOptions</code>
<code>google.maps.places.PlacesService</code>	<code>atlas.service.SearchUrl</code>

Libraries

Libraries add additional functionality to the map. Many of these libraries are in the core SDK of Azure Maps. Here are some equivalent classes to use in place of these Google Maps libraries

GOOGLE MAPS	AZURE MAPS
Drawing library	Drawing tools module
Geometry library	atlas.math
Visualization library	Heat map layer

Clean up resources

No resources to be cleaned up.

Next steps

Learn more about migrating to Azure Maps:

[Migrate a web service](#)

Tutorial: Migrate an Android app from Google Maps

3/5/2021 • 20 minutes to read • [Edit Online](#)

The Azure Maps Android SDK has an API interface that is similar to the Web SDK. If you've developed with one of these SDKs, many of the same concepts, best practices, and architectures apply. In this tutorial, you will learn how to:

- Load a map
- Localize a map
- Add markers, polylines, and polygons.
- Overlay a tile layer
- Show traffic data

All examples are provided in Java; however, you can use Kotlin with the Azure Maps Android SDK.

For more information on developing with the Android SDK by Azure Maps, see the [How-to guides for the Azure Maps Android SDK](#).

Prerequisites

1. Create an Azure Maps account by signing into the [Azure portal](#). If you don't have an Azure subscription, create a [free account](#) before you begin.
2. [Make an Azure Maps account](#)
3. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

Load a map

Loading a map in an Android app using Google or Azure Maps consists of similar steps. When using either SDK, you must:

- Get an API or subscription key to access either platform.
- Add some XML to an Activity to specify where the map should be rendered and how it should be laid out.
- Override all the life-cycle methods from the Activity containing the map view to the corresponding methods in the map class. In particular, you must override the following methods:
 - `onCreate(Bundle)`
 - `onStart()`
 - `onResume()`
 - `onPause()`
 - `onStop()`
 - `onDestroy()`
 - `onSaveInstanceState(Bundle)`
 - `onLowMemory()`
- Wait for the map to be ready before trying to access and program it.

Before: Google Maps

To display a map using the Google Maps SDK for Android, the following steps would be done:

1. Ensure Google Play services is installed.
2. Add a dependency for the Google Maps service to the module's `gradle.build` file:

```
implementation 'com.google.android.gms:play-services-maps:17.0.0'
```

3. Add a Google Maps API key inside the application section of the `google_maps_api.xml` file:

```
<meta-data android:name="com.google.android.geo.API_KEY" android:value="YOUR_GOOGLE_MAPS_KEY"/>
```

4. Add a map fragment to the main activity:

```
<com.google.android.gms.maps.MapView  
    android:id="@+id/myMap"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"/>
```

5. In the `MainActivity.java` file, you will need to import the Google Maps SDK. Forward all the life-cycle methods from the activity containing the map view to the corresponding ones in map class. Retrieve a `MapView` instance from the map fragment using the `getMapAsync(OnMapReadyCallback)` method. The `MapView` automatically initializes the maps system and the view. Edit the `MainActivity.java` file as follows:

```
import com.google.android.gms.maps.GoogleMap;  
import com.google.android.gms.maps.MapView;  
import com.google.android.gms.maps.OnMapReadyCallback;  
  
import android.support.v7.app.AppCompatActivity;  
import android.os.Bundle;  
  
public class MainActivity extends AppCompatActivity implements OnMapReadyCallback {  
  
    MapView mapView;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        mapView = findViewById(R.id.myMap);  
  
        mapView.onCreate(savedInstanceState);  
        mapView.getMapAsync(this);  
    }  
  
    @Override  
  
    public void onMapReady(GoogleMap map) {  
        //Add your post map load code here.  
    }  
  
    @Override  
    public void onResume() {  
        super.onResume();  
        mapView.onResume();  
    }  
  
    @Override  
    protected void onStart(){  
        super.onStart();  
        mapView.onStart();  
    }
```

```
}

@Override
public void onPause() {
    super.onPause();
    mapView.onPause();
}

@Override
public void onStop() {
    super.onStop();
    mapView.onStop();
}

@Override
public void onLowMemory() {
    super.onLowMemory();
    mapView.onLowMemory();
}

@Override
protected void onDestroy() {
    super.onDestroy();
    mapView.onDestroy();
}

@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    mapView.onSaveInstanceState(outState);
}
}
```

5. In the **MainActivity.kt** file, you will need to import the Google Maps SDK. Forward all the life-cycle methods from the activity containing the map view to the corresponding ones in map class. Retrieve a `MapView` instance from the map fragment using the `getMapAsync(OnMapReadyCallback)` method. The `MapView` automatically initializes the maps system and the view. Edit the **MainActivity.kt** file as follows:

```
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapView;
import com.google.android.gms.maps.OnMapReadyCallback;

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() implements OnMapReadyCallback {

    var mapView: MapView? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        mapView = findViewById(R.id.myMap)

        mapView?.onCreate(savedInstanceState)
        mapView?.getMapAsync(this)
    }

    public fun onMapReady(GoogleMap map) {
        //Add your post map load code here.
    }

    public override fun onStart() {
        super.onStart()
        mapView?.onStart()
    }

    public override fun onResume() {
        super.onResume()
        mapView?.onResume()
    }

    public override fun onPause() {
        mapView?.onPause()
        super.onPause()
    }

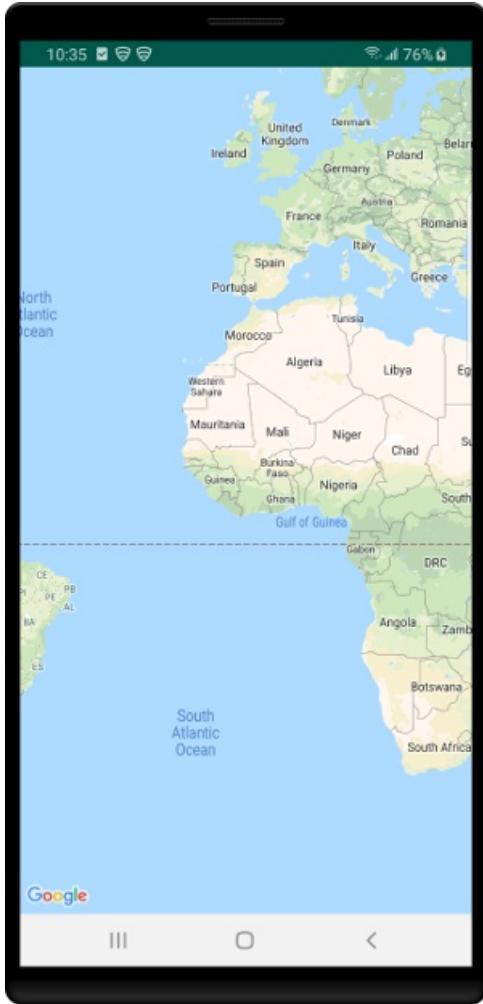
    public override fun onStop() {
        mapView?.onStop()
        super.onStop()
    }

    override fun onLowMemory() {
        mapView?.onLowMemory()
        super.onLowMemory()
    }

    override fun onDestroy() {
        mapView?.onDestroy()
        super.onDestroy()
    }

    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        mapView?.onSaveInstanceState(outState)
    }
}
```

When you run an application, the map control loads as in the following image.



After: Azure Maps

To display a map using the Azure Maps SDK for Android, the following steps need to be done:

1. Open the top-level `build.gradle` file and add the following code to the `all projects` block section:

```
maven {  
    url "https://atlas.microsoft.com/sdk/android"  
}
```

2. Update your `app/build.gradle` and add the following code to it:

- a. Make sure that your project's `minSdkVersion` is at API 21 or higher.
- b. Add the following code to the Android section:

```
compileOptions {  
    sourceCompatibility JavaVersion.VERSION_1_8  
    targetCompatibility JavaVersion.VERSION_1_8  
}
```

- c. Update your dependencies block. Add a new implementation dependency line for the latest Azure Maps Android SDK:

```
implementation "com.microsoft.azure.maps:mapcontrol:0.7"
```

NOTE

You can set the version number to "0+" to have your code always point to the latest version.

- d. Go to **File** in the toolbar and then click on **Sync Project with Gradle Files**.
3. Add a map fragment to the main activity (resources pwd> layout > activity_main.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    <com.microsoft.azure.maps.mapcontrol.MapControl
        android:id="@+id/mapcontrol"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />
</FrameLayout>
```

4. In the **MainActivity.java** file you'll need to:

- Imports the Azure Maps SDK
- Set your Azure Maps authentication information
- Get the map control instance in the **onCreate** method

Set the authentication information in the `AzureMaps` class using the `setSubscriptionKey` or `setAadProperties` methods. This global update, ensure that you add your authentication information to every view.

The map control contains its own lifecycle methods for managing Android's OpenGL lifecycle. These methods must be called directly from the contained Activity. To correctly call the map control's lifecycle methods, you must override the following lifecycle methods in the Activity that contains the map control. Call the respective map control method.

- `onCreate(Bundle)`
- `onStart()`
- `onResume()`
- `onPause()`
- `onStop()`
- `onDestroy()`
- `onSaveInstanceState(Bundle)`
- `onLowMemory()`

Edit the **MainActivity.java** file as follows:

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;
import com.microsoft.azure.maps.mapcontrol.AzureMaps;
import com.microsoft.azure.maps.mapcontrol.MapControl;
import com.microsoft.azure.maps.mapcontrol.layer.SymbolLayer;
import com.microsoft.azure.maps.mapcontrol.options.MapStyle;
import com.microsoft.azure.maps.mapcontrol.source.DataSource;
```

```
public class MainActivity extends AppCompatActivity {

    static {
        AzureMaps.setSubscriptionKey("<Your Azure Maps subscription key>");

        //Alternatively use Azure Active Directory authenticate.
        //AzureMaps.setAadProperties("<Your aad clientId>", "<Your aad AppId>", "<Your aad Tenant>");
    }

    MapControl mapControl;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mapControl = findViewById(R.id.mapcontrol);

        mapControl.onCreate(savedInstanceState);

        //Wait until the map resources are ready.
        mapControl.onReady(map -> {
            //Add your post map load code here.

        });
    }

    @Override
    public void onResume() {
        super.onResume();
        mapControl.onResume();
    }

    @Override
    protected void onStart(){
        super.onStart();
        mapControl.onStart();
    }

    @Override
    public void onPause() {
        super.onPause();
        mapControl.onPause();
    }

    @Override
    public void onStop() {
        super.onStop();
        mapControl.onStop();
    }

    @Override
    public void onLowMemory() {
        super.onLowMemory();
        mapControl.onLowMemory();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        mapControl.onDestroy();
    }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        mapControl.onSaveInstanceState(outState);
    }
}
```

4. In the **MainActivity.kt** file you'll need to:

- Imports the Azure Maps SDK
- Set your Azure Maps authentication information
- Get the map control instance in the **onCreate** method

Set the authentication information in the `AzureMaps` class using the `setSubscriptionKey` or `setAadProperties` methods. This global update, ensure that you add your authentication information to every view.

The map control contains its own lifecycle methods for managing Android's OpenGL lifecycle. These methods must be called directly from the contained Activity. To correctly call the map control's lifecycle methods, you must override the following lifecycle methods in the Activity that contains the map control. Call the respective map control method.

- `onCreate(Bundle)`
- `onStart()`
- `onResume()`
- `onPause()`
- `onStop()`
- `onDestroy()`
- `onSaveInstanceState(Bundle)`
- `onLowMemory()`

Edit the **MainActivity.kt** file as follows:

```
package com.example.myapplication

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import com.microsoft.azure.maps.mapcontrol.AzureMap
import com.microsoft.azure.maps.mapcontrol.AzureMaps
import com.microsoft.azure.maps.mapcontrol.MapControl
import com.microsoft.azure.maps.mapcontrol.events.OnReady

class MainActivity : AppCompatActivity() {

    companion object {
        init {
            AzureMaps.setSubscriptionKey("<Your Azure Maps subscription key>");

            //Alternatively use Azure Active Directory authenticate.
            //AzureMaps.setAadProperties("<Your aad clientId>", "<Your aad AppId>", "<Your aad Tenant>");
        }
    }

    var mapControl: MapControl? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        mapControl = findViewById(R.id.mapcontrol)

        mapControl?.onCreate(savedInstanceState)

        //Wait until the map resources are ready.
        mapControl?.onReady(OnReady { map: AzureMap -> })
    }
}
```

```
public override fun onStart() {
    super.onStart()
    mapControl?.onStart()
}

public override fun onResume() {
    super.onResume()
    mapControl?.onResume()
}

public override fun onPause() {
    mapControl?.onPause()
    super.onPause()
}

public override fun onStop() {
    mapControl?.onStop()
    super.onStop()
}

override fun onLowMemory() {
    mapControl?.onLowMemory()
    super.onLowMemory()
}

override fun onDestroy() {
    mapControl?.onDestroy()
    super.onDestroy()
}

override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    mapControl?.onSaveInstanceState(outState)
}
}
```

If you run your application, the map control will load as in the following image.



Notice that the Azure Maps control supports zooming out more and provides more of a world view.

TIP

If you are using an Android emulator on a Windows machine, the map may not render due to conflicts with OpenGL and software accelerated graphics rendering. The following has worked, for some people, to resolve this issue. Open the AVD Manager and select the virtual device to edit. Scroll down in the **Verify Configuration** panel. In the **Emulated Performance** section, set the **Graphics** option to **Hardware**.

Localizing the map

Localization is important if your audience is spread across multiple countries/regions or speak different languages.

Before: Google Maps

Add the following code to the `onCreate` method to set the language of the map. The code must be added before setting the context view of the map. The "fr" language code limits the language to French.

```
String languageToLoad = "fr";
Locale locale = new Locale(languageToLoad);
Locale.setDefault(locale);

Configuration config = new Configuration();
config.locale = locale;

getBaseContext().getResources().updateConfiguration(config,
    getBaseContext().getResources().getDisplayMetrics());
```

```

val languageToLoad = "fr"
val locale = Locale(languageToLoad)
Locale.setDefault(locale)

val config = Configuration()
config.locale = locale

baseContext.resources.updateConfiguration(
    config,
    baseContext.resources.displayMetrics
)

```

Here is an example of Google Maps with the language set to "fr".



After: Azure Maps

Azure Maps provides three different ways to set the language and the regional view of the map. The first option is to pass the language and regional view information to the `AzureMaps` class. This option uses the static `setLanguage` and `setView` methods globally. Meaning, the default language and regional view are set across all Azure Maps controls loaded in your app. This example sets French using the "fr-FR" language code.

```

static {
    //Set your Azure Maps Key.
    AzureMaps.setSubscriptionKey("<Your Azure Maps Key>");

    //Set the language to be used by Azure Maps.
    AzureMaps.setLanguage("fr-FR");

    //Set the regional view to be used by Azure Maps.
    AzureMaps.setView("Auto");
}

```

```

companion object {
    init {
        //Set your Azure Maps Key.
        AzureMaps.setSubscriptionKey("<Your Azure Maps Key>");

        //Set the language to be used by Azure Maps.
        AzureMaps.setLanguage("fr-FR");

        //Set the regional view to be used by Azure Maps.
        AzureMaps.setView("Auto");
    }
}

```

The second option is to pass the language and view information to the map control XML code.

```

<com.microsoft.azure.maps.mapcontrol.MapControl
    android:id="@+id/myMap"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:mapcontrol_language="fr-FR"
    app:mapcontrol_view="Auto"
/>

```

The third option is to program the language and regional map view using the maps `setStyle` method. This option updates the language and regional view anytime the code executes.

```

mapControl.onReady(map -> {
    map.setStyle(
        language("fr-FR"),
        view("Auto")
    );
});

```

```

mapControl!! .onReady { map: AzureMap ->
    map.setStyle(
        language("fr-FR"),
        view("Auto")
    )
}

```

Here is an example of Azure Maps with the language set to "fr-FR".



Review the complete list of [Supported languages](#).

Setting the map view

Dynamic maps in both Azure Maps and Google Maps can be programmatically moved to new geographic locations by calling the appropriate methods. Let's make the map display satellite aerial imagery, center the map over a location with coordinates, and change the zoom level. For this example, we'll use latitude: 35.0272, longitude: -111.0225, and zoom level of 15.

Before: Google Maps

The camera of Google Maps map control can be programmatically moved using the `moveCamera` method. The `moveCamera` method allows you to specify the center of the map and a zoom level. The `setMapType` method changes the type of map to displayed.

```
@Override  
public void onMapReady(GoogleMap googleMap) {  
    mapView = googleMap;  
  
    mapView.moveCamera(CameraUpdateFactory.newLatLngZoom(new LatLng(35.0272, -111.0225), 15));  
    mapView.setMapType(GoogleMap.MAP_TYPE_SATELLITE);  
}
```

```
public override fun onMapReady(googleMap: GoogleMap) {  
    mapView = googleMap  
  
    mapView.moveCamera(CameraUpdateFactory.newLatLngZoom(LatLng(35.0272, -111.0225), 15))  
    mapView.setMapType(GoogleMap.MAP_TYPE_SATELLITE)  
}
```



NOTE

Google Maps uses tiles that are 256 pixels in dimensions while Azure Maps uses a larger 512 pixel tile. This reduces the number of network requests needed by Azure Maps to load the same map area as Google Maps. To achieve that same viewable area as a map in Google Maps, you need to subtract the zoom level used in Google Maps by one when using Azure Maps.

After: Azure Maps

As noted previously, to achieve the same viewable area in Azure Maps subtract the zoom level used in Google Maps by one. In this case, use a zoom level of 14.

The initial map view can be set in XML attributes on the map control.

```
<com.microsoft.azure.maps.mapcontrol.MapControl  
    android:id="@+id/myMap"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    app:mapcontrol_cameraLat="35.0272"  
    app:mapcontrol_cameraLng="-111.0225"  
    app:mapcontrol_zoom="14"  
    app:mapcontrol_style="satellite"  
/>
```

The map view can be programmed using the maps `setCamera` and `setStyle` methods.

```
mapControl.onReady(map -> {  
    //Set the camera of the map.  
    map.setCamera(center(Point.fromLngLat(-111.0225, 35.0272)), zoom(14));  
  
    //Set the style of the map.  
    map.setStyle(style(MapStyle.SATELLITE));  
});
```

```
mapControl!!.onReady { map: AzureMap ->  
    //Set the camera of the map.  
    map.setCamera(center(Point.fromLngLat(-111.0225, 35.0272)), zoom(14))  
  
    //Set the style of the map.  
    map.setStyle(style(MapStyle.SATELLITE))  
}
```



Additional resources:

- Supported map styles

Adding a marker

Point data is often rendered using an image on the map. These images are referred to as markers, pushpins, pins, or symbols. The following examples render point data as markers on the map at latitude: 51.5, longitude: -0.2.

Before: Google Maps

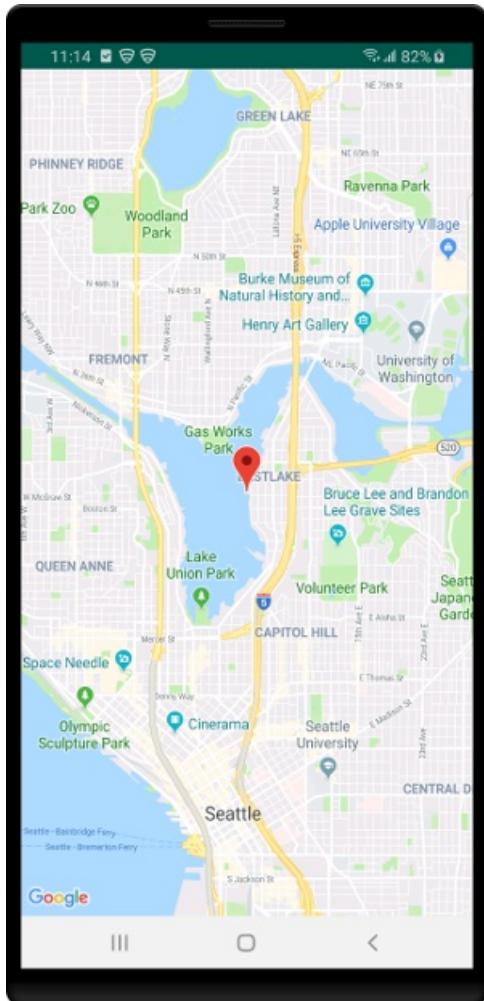
With Google Maps, markers are added using the maps `addMarker` method.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mapView = googleMap;

    mapView.addMarker(new MarkerOptions().position(new LatLng(47.64, -122.33)));
}
```

```
public override fun onMapReady(googleMap: GoogleMap) {
    mapView = googleMap

    mapView.addMarker(MarkerOptions().position(LatLng(47.64, -122.33)))
}
```



After: Azure Maps

In Azure Maps, render point data on the map by first adding the data to a data source. Then, connecting that data source to a symbol layer. The data source optimizes the management of spatial data in the map control. The

symbol layer specifies how to render point data using as an image or text.

```
mapControl.onReady(map -> {
    //Create a data source and add it to the map.
    DataSource source = new DataSource();
    map.sources.add(source);

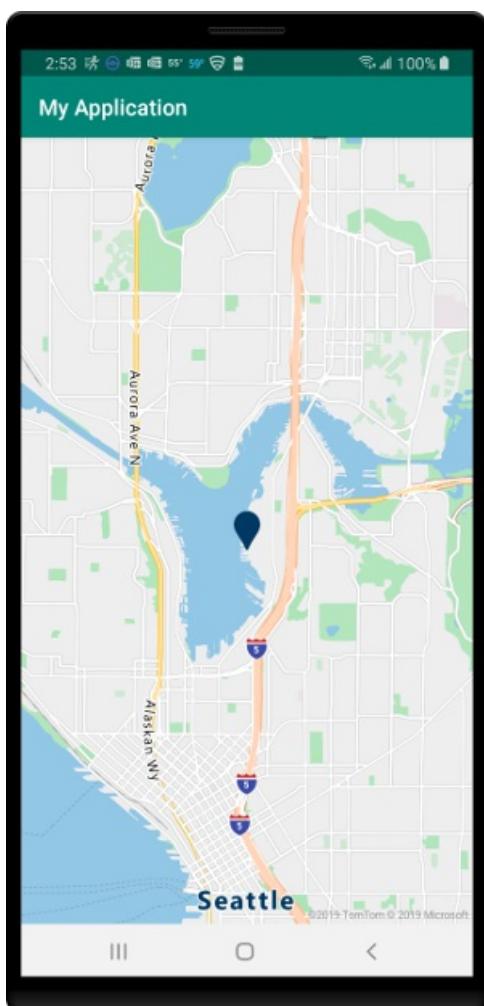
    //Create a point feature and add it to the data source.
    source.add(Feature.fromGeometry(Point.fromLngLat(-122.33, 47.64)));

    //Create a symbol layer and add it to the map.
    map.layers.add(new SymbolLayer(source));
});
```

```
mapControl!! .onReady { map: AzureMap ->
    //Create a data source and add it to the map.
    val source = new DataSource()
    map.sources.add(source)

    //Create a point feature and add it to the data source.
    source.add(Feature.fromGeometry(Point.fromLngLat(-122.33, 47.64)))

    //Create a symbol layer and add it to the map.
    map.layers.add(SymbolLayer(source))
}
```



Adding a custom marker

Custom images can be used to represent points on a map. The map in examples below uses a custom image to

display a point on the map. The point is at latitude: 51.5 and longitude: -0.2. The anchor offsets the position of the marker, so that the point of the pushpin icon aligns with the correct position on the map.



yellow-pushpin.png

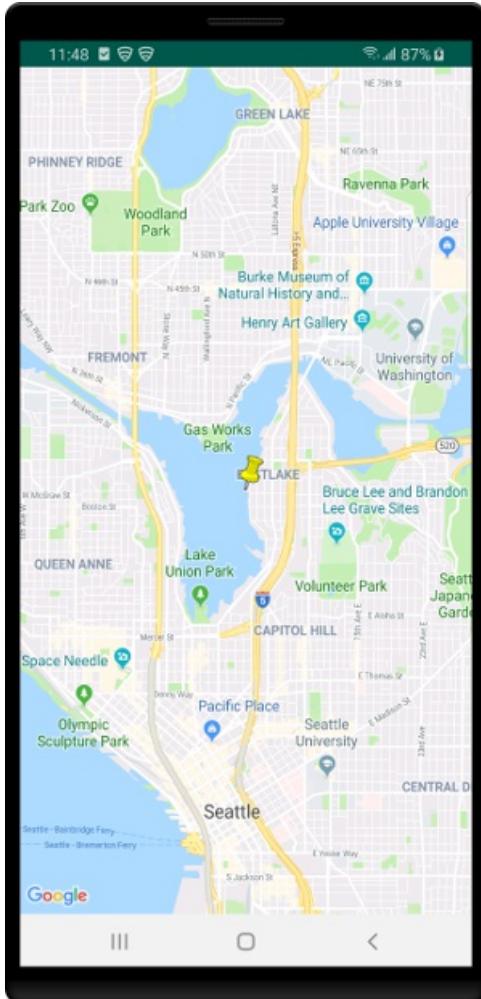
In both examples, the above image is added to the drawable folder of the apps resources.

Before: Google Maps

With Google Maps, custom images can be used for markers. Load custom images using the marker's `icon` option. To align the point of the image to the coordinate, use the `anchor` option. The anchor is relative to the dimensions of the image. In this case, the anchor is 0.2 units wide, and 1 unit high.

```
@Override  
public void onMapReady(GoogleMap googleMap) {  
    mapView = googleMap;  
  
    mapView.addMarker(new MarkerOptions().position(new LatLng(47.64, -122.33))  
        .icon(BitmapDescriptorFactory.fromResource(R.drawable.yellow-pushpin))  
        .anchor(0.2f, 1f));  
}
```

```
public override fun onMapReady(googleMap: GoogleMap) {  
    mapView = googleMap;  
  
    mapView.addMarker(MarkerOptions().position(LatLng(47.64, -122.33))  
        .icon(BitmapDescriptorFactory.fromResource(R.drawable.yellow-pushpin))  
        .anchor(0.2f, 1f))  
}
```



After: Azure Maps

Symbol layers in Azure Maps support custom images, but first, the image needs to be loaded to the map resources and assigned a unique ID. Then, the symbol layer needs to reference this ID. Offset the symbol to align to the correct point on the image using the `iconOffset` option. The icon offset is in pixels. By default, the offset is relative to the bottom-center of the image, but this offset value can be adjusted using the `iconAnchor` option. This example sets the `iconAnchor` option to `"center"`. It uses an icon offset to move the image five pixels to the right and 15 pixels up to align with the point of the pushpin image.

```
mapControl.onReady(map => {
    //Create a data source and add it to the map.
    DataSource source = new DataSource();
    map.sources.add(source);

    //Create a point feature and add it to the data source.
    source.add(Feature.fromGeometry(Point.fromLngLat(-122.33, 47.64)));

    //Load the custom image icon into the map resources.
    map.images.add("my-yellow-pin", R.drawable.yellow_pushpin);

    //Create a symbol that uses the custom image icon and add it to the map.
    map.layers.add(new SymbolLayer(source,
        iconImage("my-yellow-pin"),
        iconAnchor(AnchorType.CENTER),
        iconOffset(new Float[]{5f, -15f})));
});
```

```

mapControl!!.onReady { map: AzureMap ->
    //Create a data source and add it to the map.
    val source = DataSource()
    map.sources.add(source)

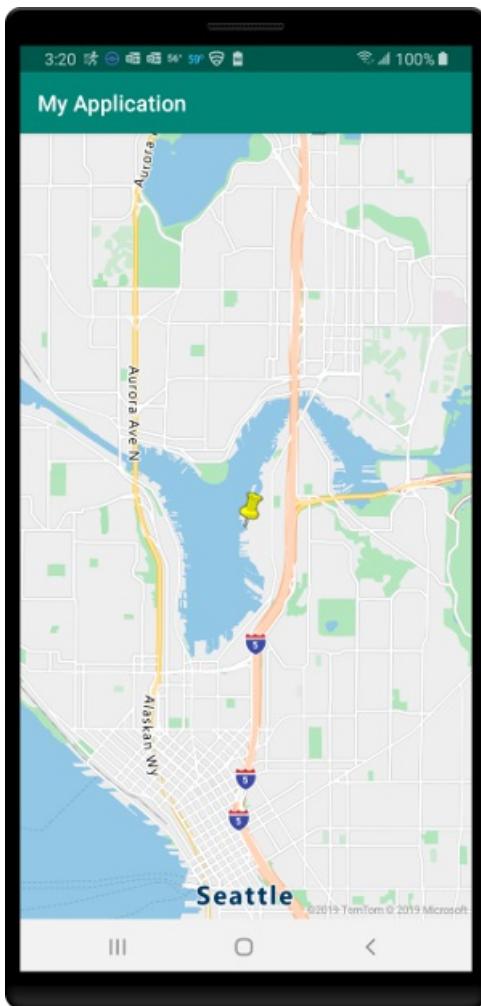
    //Create a point feature and add it to the data source.
    source.add(Feature.fromGeometry(Point.fromLngLat(-122.33, 47.64)))

    //Load the custom image icon into the map resources.
    map.images.add("my-yellow-pin", R.drawable.yellow_pushpin)

    //Create a symbol that uses the custom image icon and add it to the map.
    map.layers.add(SymbolLayer(source,
        iconImage("my-yellow-pin"),
        iconAnchor(AnchorType.CENTER),
        iconOffset(arrayOf(0f, -1.5f))))}

}

```



Adding a polyline

Polylines are used to represent a line or path on the map. The following examples show how to create a dashed polyline on the map.

Before: Google Maps

With Google Maps, render a polyline using the `PolylineOptions` class. Add the polyline to the map using the `addPolyline` method. Set the stroke color using the `color` option. Set the stroke width using the `width` option. Add a stroke dash array using the `pattern` option.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mapView = googleMap;

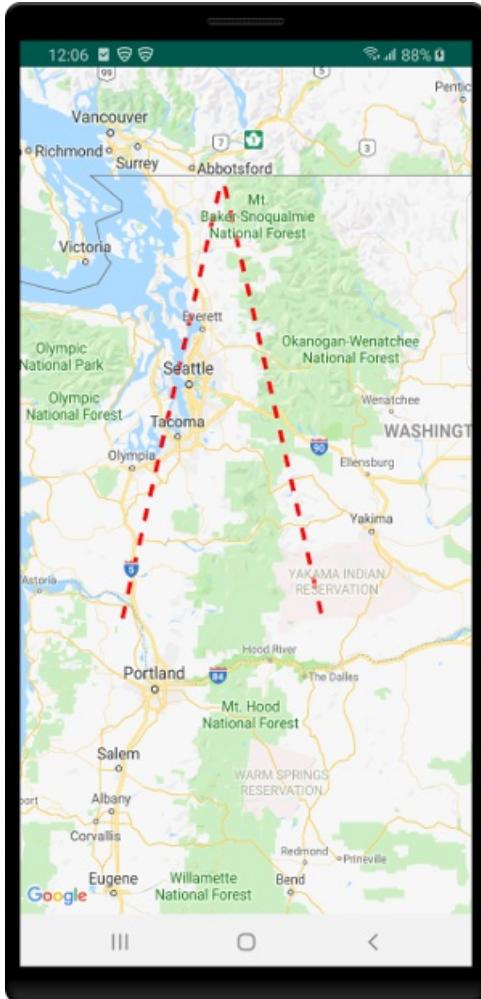
    //Create the options for the polyline.
    PolylineOptions lineOptions = new PolylineOptions()
        .add(new LatLng(46, -123))
        .add(new LatLng(49, -122))
        .add(new LatLng(46, -121))
        .color(Color.RED)
        .width(10f)
        .pattern(Arrays.<PatternItem>asList(
            new Dash(30f), new Gap(30f)));

    //Add the polyline to the map.
    Polyline polyline = mapView.addPolyline(lineOptions);
}
```

```
public override fun onMapReady(googleMap: GoogleMap) {
    mapView = googleMap

    //Create the options for the polyline.
    val lineOptions = new PolylineOptions()
        .add(new LatLng(46, -123))
        .add(new LatLng(49, -122))
        .add(new LatLng(46, -121))
        .color(Color.RED)
        .width(10f)
        .pattern(Arrays.<PatternItem>asList(
            new Dash(30f), new Gap(30f)))

    //Add the polyline to the map.
    val polyline = mapView.addPolyline(lineOptions)
}
```



After: Azure Maps

In Azure Maps, polylines are called `LineString` or `MultiLineString` objects. Add these objects to a data source and render them using a line layer. Set the stroke width using the `strokeWidth` option. Add a stroke dash array using the `strokeDashArray` option.

The stroke width and the dash array "pixel" units in the Azure Maps Web SDK, is the same as in the Google Maps service. Both accept the same values to produce the same results.

```
mapControl.onReady(map => {
    //Create a data source and add it to the map.
    DataSource source = new DataSource();
    map.sources.add(source);

    //Create an array of points.
    List<Point> points = Arrays.asList(
        Point.fromLngLat(-123, 46),
        Point.fromLngLat(-122, 49),
        Point.fromLngLat(-121, 46));

    //Create a LineString feature and add it to the data source.
    source.add(Feature.fromGeometry(LineString.fromLngLats(points)));

    //Create a line layer and add it to the map.
    map.layers.add(new LineLayer(source,
        strokeColor("red"),
        strokeWidth(4f),
        strokeDashArray(new Float[]{3f, 3f})));
});
```

```

mapControl!! .onReady { map: AzureMap ->
    //Create a data source and add it to the map.
    val source = DataSource()
    map.sources.add(source)

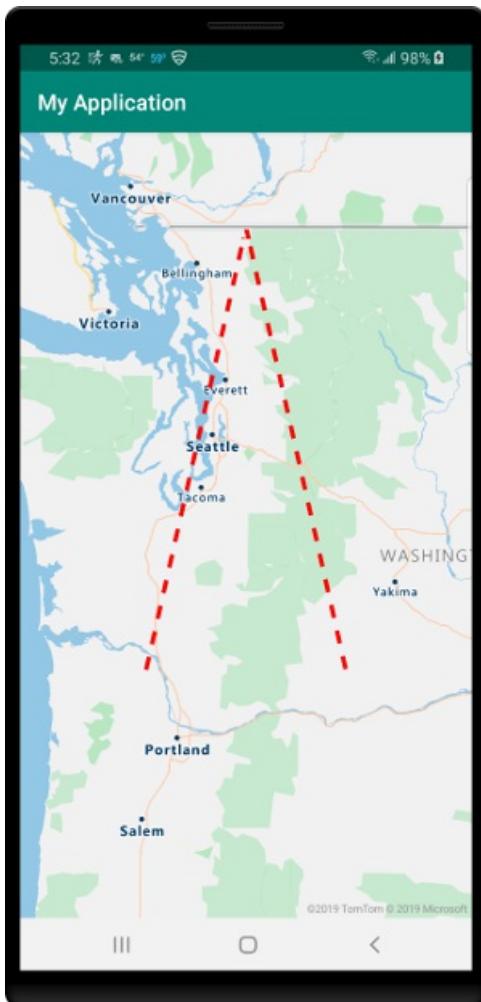
    //Create an array of points.
    val points = Arrays.asList(
        Point.fromLngLat(-123, 46),
        Point.fromLngLat(-122, 49),
        Point.fromLngLat(-121, 46))

    //Create a LineString feature and add it to the data source.
    source.add(Feature.fromGeometry(LineString.fromLngLats(points)))

    //Create a line layer and add it to the map.
    map.layers.add(LineLayer(source,
        strokeColor("red"),
        strokeWidth(4f),
        strokeDashArray(new Float[]{3f, 3f})))
}

}

```



Adding a polygon

Polygons are used to represent an area on the map. The next examples show you how to create a polygon. This polygon forms a triangle based on the center coordinate of the map.

Before: Google Maps

With Google Maps, render a polygon using the `PolygonOptions` class. Add the polygon to the map using the `addPolygon` method. Set the fill and stroke colors using the `fillColor` and `strokeColor` options, respectively. Set the stroke width using the `strokeWidth` option.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mapView = googleMap;

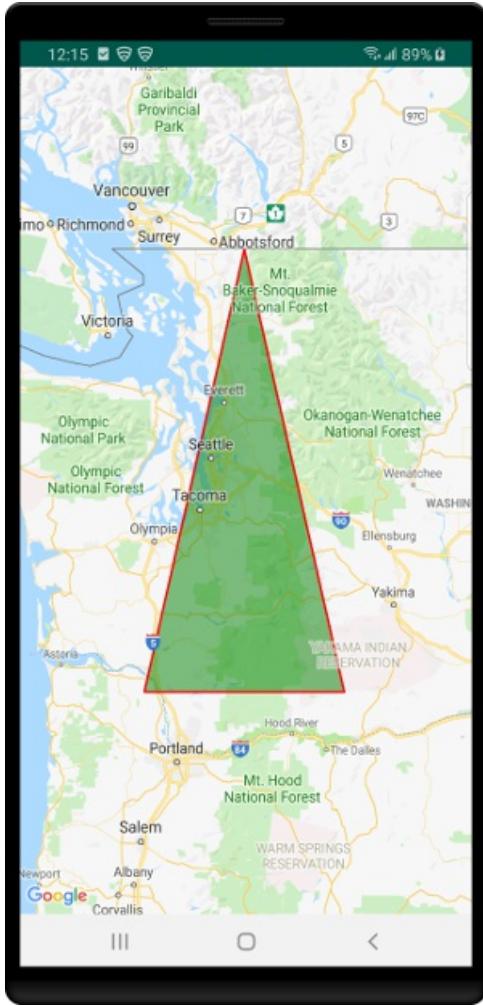
    //Create the options for the polygon.
    PolygonOptions polygonOptions = new PolygonOptions()
        .add(new LatLng(46, -123))
        .add(new LatLng(49, -122))
        .add(new LatLng(46, -121))
        .add(new LatLng(46, -123)) //Close the polygon.
        .fillColor(Color.argb(128, 0, 128, 0))
        .strokeColor(Color.RED)
        .strokeWidth(5f);

    //Add the polygon to the map.
    Polygon polygon = mapView.addPolygon(polygonOptions);
}
```

```
public override fun onMapReady(googleMap: GoogleMap) {
    mapView = googleMap;

    //Create the options for the polygon.
    val polygonOptions = PolygonOptions()
        .add(new LatLng(46, -123))
        .add(new LatLng(49, -122))
        .add(new LatLng(46, -121))
        .add(new LatLng(46, -123)) //Close the polygon.
        .fillColor(Color.argb(128, 0, 128, 0))
        .strokeColor(Color.RED)
        .strokeWidth(5f)

    //valAdd the polygon to the map.
    Polygon polygon = mapView.addPolygon(polygonOptions)
}
```



After: Azure Maps

In Azure Maps, add `Polygon` and `MultiPolygon` objects to a data source and render them on the map using layers. Render the area of a polygon in a polygon layer. Render the outline of a polygon using a line layer. Set the stroke color and width using the `strokeColor` and `strokeWidth` options.

The stroke width and dash array "pixel" units in Azure Maps Web SDK align with the respective units in Google Maps. Both accept the same values and produce the same results.

```

mapControl.onReady(map -> {
    //Create a data source and add it to the map.
    DataSource source = new DataSource();
    map.sources.add(source);

    //Create an array of point arrays to create polygon rings.
    List<List<Point>> rings = Arrays.asList(Arrays.asList(
        Point.fromLngLat(-123, 46),
        Point.fromLngLat(-122, 49),
        Point.fromLngLat(-121, 46),
        Point.fromLngLat(-123, 46)));

    //Create a Polygon feature and add it to the data source.
    source.add(Feature.fromGeometry(Polygon.fromLngLats(rings)));

    //Add a polygon layer for rendering the polygon area.
    map.layers.add(new PolygonLayer(source,
        fillColor("green"),
        fillOpacity(0.5f)));

    //Add a line layer for rendering the polygon outline.
    map.layers.add(new LineLayer(source,
        strokeColor("red"),
        strokeWidth(2f)));
});

```

```

mapControl!! .onReady { map: AzureMap ->
    //Create a data source and add it to the map.
    val source = DataSource()
    map.sources.add(source)

    //Create an array of point arrays to create polygon rings.
    val rings = Arrays.asList(Arrays.asList(
        Point.fromLngLat(-123, 46),
        Point.fromLngLat(-122, 49),
        Point.fromLngLat(-121, 46),
        Point.fromLngLat(-123, 46)))

    //Create a Polygon feature and add it to the data source.
    source.add(Feature.fromGeometry(Polygon.fromLngLats(rings)))

    //Add a polygon layer for rendering the polygon area.
    map.layers.add(PolygonLayer(source,
        fillColor("green"),
        fillOpacity(0.5f)))

    //Add a line layer for rendering the polygon outline.
    map.layers.add(LineLayer(source,
        strokeColor("red"),
        strokeWidth(2f)))
}

```



Overlay a tile layer

Use Tile layers to overlay layer images that have been broken up into smaller tiled images, which align with the maps tiling system. This approach is a common way of overlaying layer images or large data sets. Tile layers are known as Image overlays in Google Maps.

The following examples overlay a weather radar tile layer from Iowa Environmental Mesonet of Iowa State University. The tiles are 256 pixels in size.

Before: Google Maps

With Google Maps, a tile layer can be overlaid on top of the map. Use the `TileOverlayOptions` class. Add the tile layer to the map using the `addTileLayer` method. To make the tiles semi-transparent, the `transparency` option is set to 0.2, or 20% transparent.

```

@Override
public void onMapReady(GoogleMap googleMap) {
    mapView = googleMap;

    //Create the options for the tile layer.
    TileOverlayOptions tileLayer = new TileOverlayOptions()
        .tileProvider(new UrlTileProvider(256, 256) {
            @Override
            public URL getTileUrl(int x, int y, int zoom) {

                try {
                    //Define the URL pattern for the tile images.
                    return new
URL(String.format("https://mesonet.agron.iastate.edu/cache/tile.py/1.0.0/nexrad-n0q-900913/%d/%d/%d.png",
zoom, x, y));
                }catch (MalformedURLException e) {
                    throw new AssertionError(e);
                }
            }
        }).transparency(0.2f);

    //Add the tile layer to the map.
    mapView.addTileOverlay(tileLayer);
}

```

```

public override fun onMapReady(googleMap: GoogleMap) {
    mapView = googleMap
    //Create the options for the tile layer.
    val tileLayer: TileOverlayOptions = TileOverlayOptions()
        .tileProvider(object : UrlTileProvider(256, 256) {
            fun getTileUrl(x: Int, y: Int, zoom: Int): URL? {
                return try { //Define the URL pattern for the tile images.
                    URL(
                        String.format(
                            "https://mesonet.agron.iastate.edu/cache/tile.py/1.0.0/nexrad-n0q-
900913/%d/%d/%d.png",
                            zoom,
                            x,
                            y
                        )
                ) catch (e: MalformedURLException) {
                    throw AssertionError(e)
                }
            }
        }).transparency(0.2f)
    //Add the tile layer to the map.
    mapView.addTileOverlay(tileLayer)
}

```



After: Azure Maps

A tile layer can be added to the map in a similar way as any other layer. A formatted URL that has `x`, `y`, and `z` placeholders; `{x}` , `{y}` , `{z}` respectively is used to tell the layer where to access the tiles. Also, tile layers in Azure Maps support `{quadkey}` , `{bbox-epsg-3857}` , and `{subdomain}` placeholders. To make the tile layer semi-transparent, an opacity value of 0.8 is used. Opacity and transparency, although similar, use inverted values. To convert between both options, subtract their value from the number one.

TIP

In Azure Maps, it's convenient to render layers below other layers, including base map layers. Also, it's often desirable to render tile layers below the map labels so that they are easy to read. The `map.layers.add` method takes a second parameter which is the id of the layer in which to insert the new layer below. To insert a tile layer below the map labels, the following code can be used: `map.layers.add(myTileLayer, "labels");`

```
mapControl.onReady(map => {
    //Add a tile layer to the map, below the map labels.
    map.layers.add(new TileLayer(
        tileSize(256)
    ), "labels");
});
```

```
mapControl!.onReady { map: AzureMap ->
    //Add a tile layer to the map, below the map labels.
    map.layers.add(TileLayer(
        tileSize(256),
        tileUrl("https://mesonet.agron.iastate.edu/cache/tile.py/1.0.0/nexrad-n0q-900913/{z}/{x}/{y}.png"),
        opacity(0.8f)
    ), "labels")
}
```



Show traffic

Both Azure Maps and Google maps have options to overlay traffic data.

Before: Google Maps

With Google Maps, traffic flow data can be overlaid on top of the map by passing true to the map's

```
setTrafficEnabled
```

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mapView = googleMap;

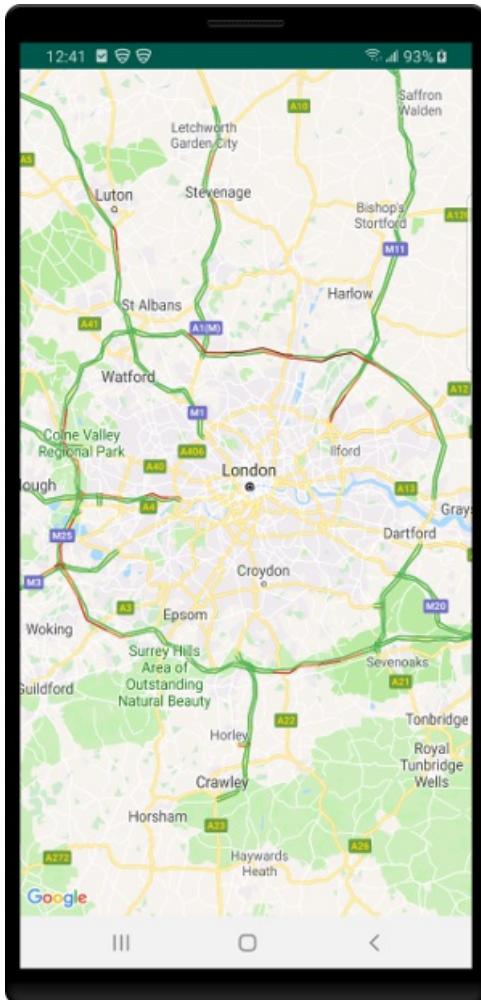
    mapView.setTrafficEnabled(true);
}
```

```

public override fun onMapReady(googleMap: GoogleMap) {
    mapView = googleMap

    mapView.setTrafficEnabled(true)
}

```



After: Azure Maps

Azure Maps provides several different options for displaying traffic. Traffic incidents, such as road closures and accidents can be displayed as icons on the map. Traffic flow and color coded roads can be overlaid on the map. The colors can be modified to appear relative to the posted speed limit, relative to the normal expected delay, or the absolute delay. Incident data in Azure Maps is updated every minute, and flow data is updated every two minutes.

```

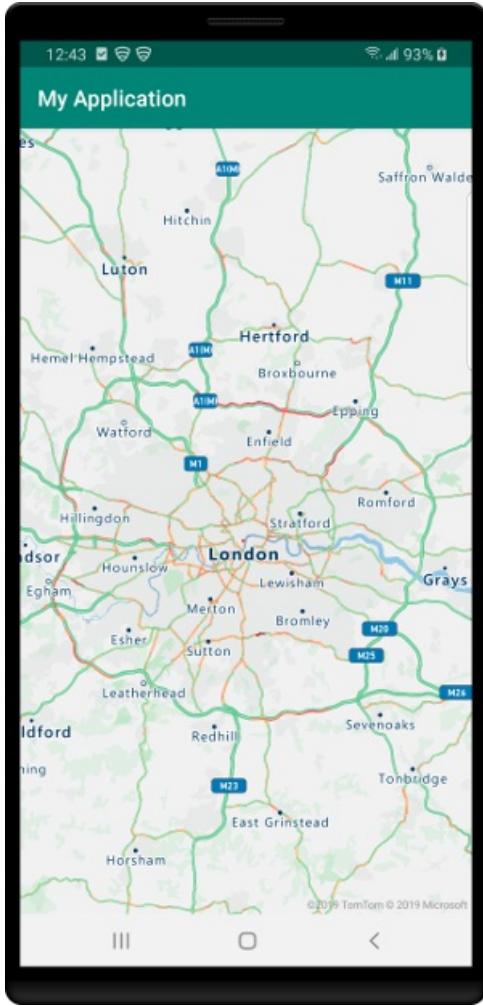
mapControl.onReady(map -> {
    map.setTraffic(
        incidents(true),
        flow(TrafficFlow.RELATIVE));
});

```

```

mapControl!! .onReady { map: AzureMap ->
    map.setTraffic(
        incidents(true),
        flow(TrafficFlow.RELATIVE))
}

```



Clean up resources

No resources to be cleaned up.

Next steps

Learn more about migrating Azure Maps:

[Migrate an Android app](#)

Tutorial: Migrate web service from Google Maps

6/8/2021 • 22 minutes to read • [Edit Online](#)

Both Azure and Google Maps provide access to spatial APIs through REST web services. The API interfaces of these platforms perform similar functionalities. But, they each use different naming conventions and response objects.

In this tutorial, you will learn how to:

- Forward and reverse geocoding
- Search for points of interest
- Calculate routes and directions
- Retrieve a map image
- Calculate a distance matrix
- Get time zone details

You will also learn:

- Which Azure Maps REST service when migrating from a Google Maps Web Service
- Tips on how to get the most out of the Azure Maps services
- Insights into other related Azure Maps services

The table shows the Azure Maps service APIs, which have a similar functionality to the listed Google Maps service APIs.

GOOGLE MAPS SERVICE API	AZURE MAPS SERVICE API
Directions	Route
Distance Matrix	Route Matrix
Geocoding	Search
Places Search	Search
Place Autocomplete	Search
Snap to Road	See Calculate routes and directions section.
Speed Limits	See Reverse geocode a coordinate section.
Static Map	Render
Time Zone	Time Zone
Elevation	Elevation

The following service APIs aren't currently available in Azure Maps:

- Geolocation - Azure Maps does have a service called Geolocation, but it provides IP Address to location information, but does not currently support cell tower or WiFi triangulation.

- Places details and photos - Phone numbers and website URL are available in the Azure Maps search API.
- Map URLs
- Nearest Roads - This is achievable using the Web SDK as shown [here](#), but not available as a service currently.
- Static street view

Azure Maps has several additional REST web services that may be of interest:

- Spatial operations:** Offload complex spatial calculations and operations, such as geofencing, to a service.
- Traffic:** Access real-time traffic flow and incident data.

Prerequisites

- Sign in to the [Azure portal](#). If you don't have an Azure subscription, create a [free account](#) before you begin.
- [Make an Azure Maps account](#)
- [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

Geocoding addresses

Geocoding is the process of converting an address into a coordinate. For example, "1 Microsoft way, Redmond, WA" converts to longitude: -122.1298, latitude: 47.64005. Then, Coordinates can be used for different kind of purposes, such as, positioning a centering a marker on a map.

Azure Maps provides several methods for geocoding addresses:

- Free-form address geocoding:** Specify a single address string and process the request immediately. "1 Microsoft way, Redmond, WA" is an example of a single address string. This API is recommended if you need to geocode individual addresses quickly.
- Structured address geocoding:** Specify the parts of a single address, such as the street name, city, country/region, and postal code and process the request immediately. This API is recommended if you need to geocode individual addresses quickly and the data is already parsed into its individual address parts.
- Batch address geocoding:** Create a request containing up to 10,000 addresses and have them processed over a period of time. All the addresses will be geocoded in parallel on the server and when completed the full result set can be downloaded. This is recommended for geocoding large data sets.
- Fuzzy search:** This API combines address geocoding with point of interest search. This API takes in a free-form string. This string can be an address, place, landmark, point of interest, or point of interest category. This API processes the request near real time. This API is recommended for applications where users search for addresses or points of interest in the same textbox.
- Fuzzy batch search:** Create a request containing up to 10,000 addresses, places, landmarks, or point of interests and have them processed over a period of time. All the data will be processed in parallel on the server and when completed the full result set can be downloaded.

The following table cross-references the Google Maps API parameters with the comparable API parameters in Azure Maps.

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>address</code>	<code>query</code>
<code>bounds</code>	<code>topLeft</code> and <code>bottomRight</code>

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
components	streetNumber streetName crossStreet postalCode municipality - city / town municipalitySubdivision – neighborhood, sub / super city countrySubdivision - state or province countrySecondarySubdivision - county countryTertiarySubdivision - district countryCode - two letter country/region code
key	subscription-key – See also the Authentication with Azure Maps documentation.
language	language – See supported languages documentation.
region	countrySet

An example of how to use the search service is documented [here](#). Be sure to review [best practices for search](#).

TIP

The free-form address geocoding and fuzzy search APIs can be used in autocomplete mode by adding `&typeahead=true` to the request URL. This will tell the server that the input text is likely partial, and the search will go into predictive mode.

Reverse geocode a coordinate

Reverse geocoding is the process of converting geographic coordinates into an approximate address. Coordinates with "longitude: -122.1298, latitude: 47.64005" convert to "1 Microsoft way, Redmond, WA".

Azure Maps provides several reverse geocoding methods:

- **Address reverse geocoder**: Specify a single geographic coordinate to get the approximate address corresponding to this coordinate. Processes the request near real time.
- **Cross street reverse geocoder**: Specify a single geographic coordinate to get nearby cross street information and process the request immediately. For example, you may receive the following cross streets 1st Ave and Main St.
- **Batch address reverse geocoder**: Create a request containing up to 10,000 coordinates and have them processed over a period of time. All data will be processed in parallel on the server. When the request completes, you can download the full set of results.

This table cross-references the Google Maps API parameters with the comparable API parameters in Azure Maps.

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
key	subscription-key – See also the Authentication with Azure Maps documentation.
language	language – See supported languages documentation.

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
latlng	query
location_type	N/A
result_type	entityType

Review [best practices for search](#).

The Azure Maps reverse geocoding API has some additional features, which aren't available in Google Maps. These features might be useful to integrate with your application, as you migrate your app:

- Retrieve speed limit data
- Retrieve road use information: local road, arterial, limited access, ramp, and so on
- Retrieve the side of street at which a coordinate is located

Search for points of interest

Point of interest data can be searched in Google Maps using the Places Search API. This API provides three different ways to search for points of interest:

- **Find place from text:** Searches for a point of interest based on its name, address, or phone number.
- **Nearby Search:** Searches for points of interests that are within a certain distance of a location.
- **Text Search:** Searches for places using a free-form text, which includes point of interest and location information. For example, "pizza in New York" or "restaurants near main st".

Azure Maps provides several search APIs for points of interest:

- **POI search:** Search for points of interests by name. For example, "Starbucks".
- **POI category search:** Search for points of interests by category. For example, "restaurant".
- **Nearby search:** Searches for points of interests that are within a certain distance of a location.
- **Fuzzy search:** This API combines address geocoding with point of interest search. This API takes in a free-form string that can be an address, place, landmark, point of interest, or point of interest category. It processes the request near real time. This API is recommended for applications where users search for addresses or points of interest in the same textbox.
- **Search within geometry:** Search for points of interests that are within a specified geometry. For example, search a point of interest within a polygon.
- **Search along route:** Search for points of interests that are along a specified route path.
- **Fuzzy batch search:** Create a request containing up to 10,000 addresses, places, landmarks, or point of interests. Processed the request over a period of time. All data will be processed in parallel on the server. When the request completes processing, you can download the full set of result.

Currently Azure Maps doesn't have a comparable API to the Text Search API in Google Maps.

TIP

The POI search, POI category search, and fuzzy search APIs can be used in autocomplete mode by adding `&typeahead=true` to the request URL. This will tell the server that the input text is likely partial. The API will conduct the search in predictive mode.

Review the [best practices for search](#) documentation.

Find place from text

Use the Azure Maps [POI search](#) and [Fuzzy search](#) to search for points of interests by name or address.

The table cross-references the Google Maps API parameters with the comparable Azure Maps API parameters.

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>fields</code>	<i>N/A</i>
<code>input</code>	<code>query</code>
<code>inputtype</code>	<i>N/A</i>
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation.
<code>language</code>	<code>language</code> – See supported languages documentation.
<code>locationbias</code>	<code>lat</code> , <code>lon</code> and <code>radius</code> <code>topLeft</code> and <code>btmRight</code> <code>countrySet</code>

Nearby search

Use the [Nearby search](#) API to retrieve nearby points of interests, in Azure Maps.

The table shows the Google Maps API parameters with the comparable Azure Maps API parameters.

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation.
<code>keyword</code>	<code>categorySet</code> and <code>brandSet</code>
<code>language</code>	<code>language</code> – See supported languages documentation.
<code>location</code>	<code>lat</code> and <code>lon</code>
<code>maxprice</code>	<i>N/A</i>
<code>minprice</code>	<i>N/A</i>
<code>name</code>	<code>categorySet</code> and <code>brandSet</code>
<code>opennow</code>	<i>N/A</i>
<code>pagetoken</code>	<code>ofs</code> and <code>limit</code>
<code>radius</code>	<code>radius</code>
<code>rankby</code>	<i>N/A</i>

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>type</code>	<code>categorySet</code> – See supported search categories documentation.

Calculate routes and directions

Calculate routes and directions using Azure Maps. Azure Maps has many of the same functionalities as the Google Maps routing service, such as:

- Arrival and departure times.
- Real-time and predictive based traffic routes.
- Different modes of transportation. Such as, driving, walking, bicycling.

NOTE

Azure Maps requires all waypoints to be coordinates. Addresses must be geocoded first.

The Azure Maps routing service provides the following APIs for calculating routes:

- **Calculate route**: Calculate a route and have the request processed immediately. This API supports both GET and POST requests. POST requests are recommended when specifying a large number of waypoints or when using lots of the route options to ensure that the URL request doesn't become too long and cause issues. The POST Route Direction in Azure Maps has an option can that take in thousands of [supporting points](#) and will use them to recreate a logical route path between them (snap to road).
- **Batch route**: Create a request containing up to 1,000 route request and have them processed over a period of time. All the data will be processed in parallel on the server and when completed the full result set can be downloaded.
- **Mobility services (Preview)**: Calculate routes and directions using public transit.

The table cross-references the Google Maps API parameters with the comparable API parameters in Azure Maps.

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
<code>alternatives</code>	<code>maxAlternatives</code>
<code>arrival_time</code>	<code>arriveAt</code>
<code>avoid</code>	<code>avoid</code>
<code>departure_time</code>	<code>departAt</code>
<code>destination</code>	<code>query</code> – coordinates in the format "lat0,lon0:lat1,lon1..."
<code>key</code>	<code>subscription-key</code> – See also the Authentication with Azure Maps documentation.
<code>language</code>	<code>language</code> – See supported languages documentation.
<code>mode</code>	<code>travelMode</code>

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
optimize	computeBestOrder
origin	query
region	N/A – This feature is geocoding related. Use the <i>countrySet</i> parameter when using the Azure Maps geocoding API.
traffic_model	N/A – Can only specify if traffic data should be used with the <i>traffic</i> parameter.
transit_mode	See Mobility services (Preview) documentation
transit_routing_preference	See Mobility services (Preview) documentation
units	N/A – Azure Maps only uses the metric system.
waypoints	query

TIP

By default, the Azure Maps route API only returns a summary. It returns the distance and times and the coordinates for the route path. Use the `instructionsType` parameter to retrieve turn-by-turn instructions. And, use the `routeRepresentation` parameter to filter out the summary and route path.

Azure Maps routing API has additional features, that aren't available in Google Maps. When migrating your app, consider using these features, you might find them useful.

- Support for route type: shortest, fastest, trilling, and most fuel efficient.
- Support for additional travel modes: bus, motorcycle, taxi, truck, and van.
- Support for 150 waypoints.
- Compute multiple travel times in a single request; historic traffic, live traffic, no traffic.
- Avoid additional road types: carpool roads, unpaved roads, already used roads.
- Specify custom areas to avoid.
- Limit the elevation, which the route may ascend.
- Route based on engine specifications. Calculate routes for combustion or electric vehicles based on engine specifications, and the remaining fuel or charge.
- Support commercial vehicle route parameters. Such as, vehicle dimensions, weight, number of axels, and cargo type.
- Specify maximum vehicle speed.

In addition to this, the route service in Azure Maps supports [calculating routable ranges](#). Calculating routable ranges is also known as isochrones. It entails generating a polygon covering an area that can be traveled to in any direction from an origin point. All under a specified amount of time or amount of fuel or charge.

Review the [best practices for routing](#) documentation.

Retrieve a map image

Azure Maps provides an API for rendering the static map images with data overlaid. The [Map image render API](#) in Azure Maps is comparable to the static map API in Google Maps.

NOTE

Azure Maps requires the center, all the marker, and the path locations to be coordinates in "longitude,latitude" format. Whereas, Google Maps uses the "latitude,longitude" format. Addresses will need to be geocoded first.

The table cross-references the Google Maps API parameters with the comparable API parameters in Azure Maps.

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
center	center
format	format – specified as part of URL path. Currently only PNG supported.
key	subscription-key – See also the Authentication with Azure Maps documentation.
language	language – See supported languages documentation.
maptype	layer and style – See Supported map styles documentation.
markers	pins
path	path
region	N/A – This is a geocoding related feature. Use the countrySet parameter when using the Azure Maps geocoding API.
scale	N/A
size	width and height – can be up to 8192x8192 in size.
style	N/A
visible	N/A
zoom	zoom

NOTE

In the Azure Maps tile system, tiles are twice the size of map tiles used in Google Maps. As such the zoom level value in Azure Maps will appear one zoom level closer in Azure Maps compared to Google Maps. To compensate for this difference, decrement the zoom level in the requests you are migrating.

For more information, see the [How-to guide on the map image render API](#).

In addition to being able to generate a static map image, the Azure Maps render service provides the ability to directly access map tiles in raster (PNG) and vector format:

- **Map tile:** Retrieve raster (PNG) and vector tiles for the base maps (roads, boundaries, background).

- **Map imagery tile**: Retrieve aerial and satellite imagery tiles.

TIP

Many Google Maps applications where switched from interactive map experiences to static map images a few years ago. This was done as a cost saving method. In Azure Maps, it is usually more cost effective to use the interactive map control in the Web SDK. The interactive map control charges based the number of tile loads. Map tiles in Azure Maps are large. Often, it takes only a few tiles to recreate the same map view as a static map. Map tiles are cached automatically by the browser. As such, the interactive map control often generates a fraction of a transaction when reproducing a static map view. Panning and zooming will load more tiles; however, there are options in the map control to disable this behavior. The interactive map control also provides a lot more visualization options than the static map services.

Marker URL parameter format comparison

Before: Google Maps

Add markers using the `markers` parameter in the URL. The `markers` parameter takes in a style and a list of locations to be rendered on the map with that style as shown below:

```
&markers=markerStyles|markerLocation1|markerLocation2|...
```

To add additional styles, use the `markers` parameters to the URL with a different style and set of locations.

Specify marker locations with the "latitude,longitude" format.

Add marker styles with the `optionName:value` format, with multiple styles separated by pipe (|) characters like this "optionName1:value1|optionName2:value2". Note the option names and values are separated with a colon (:). Use the following names of style option to style markers in Google Maps:

- `color` – The color of the default marker icon. Can be a 24-bit hex color (`0xrrggbb`) or one of the following values; `black`, `brown`, `green`, `purple`, `yellow`, `blue`, `gray`, `orange`, `red`, `white`.
- `label` – A single uppercase alphanumeric character to display on top of the icon.
- `size` - The size of the marker. Can be `tiny`, `mid`, or `small`.

Use the following style options names for Custom icons in Google Maps:

- `anchor` – Specifies how to align the icon image to the coordinate. Can be a pixel (x,y) value or one of the following values; `top`, `bottom`, `left`, `right`, `center`, `topleft`, `topright`, `bottomleft`, or `bottomright`.
- `icon` – A URL pointing to the icon image.

For example, let's add a red, mid-sized marker to the map at longitude: -110, latitude: 45:

```
&markers=color:red|size:mid|45,-110
```



After: Azure Maps

Add markers to a static map image by specifying the `pins` parameter in the URL. Like Google Maps, specify a style and a list of locations in the parameter. The `pins` parameter can be specified multiple times to support markers with different styles.

```
&pins=iconType|pinStyles||pinLocation1|pinLocation2|...
```

To use additional styles, add additional `pins` parameters to the URL with a different style and set of locations.

In Azure Maps, the pin location needs to be in the "longitude latitude" format. Google Maps uses "latitude,longitude" format. A space, not a comma, separates longitude and latitude in the Azure Maps format.

The `iconType` specifies the type of pin to create. It can have the following values:

- `default` – The default pin icon.
- `none` – No icon is displayed, only labels will be rendered.
- `custom` – Specifies a custom icon is to be used. A URL pointing to the icon image can be added to the end of the `pins` parameter after the pin location information.
- `{udid}` – A Unique Data ID (UDID) for an icon stored in the Azure Maps Data Storage platform.

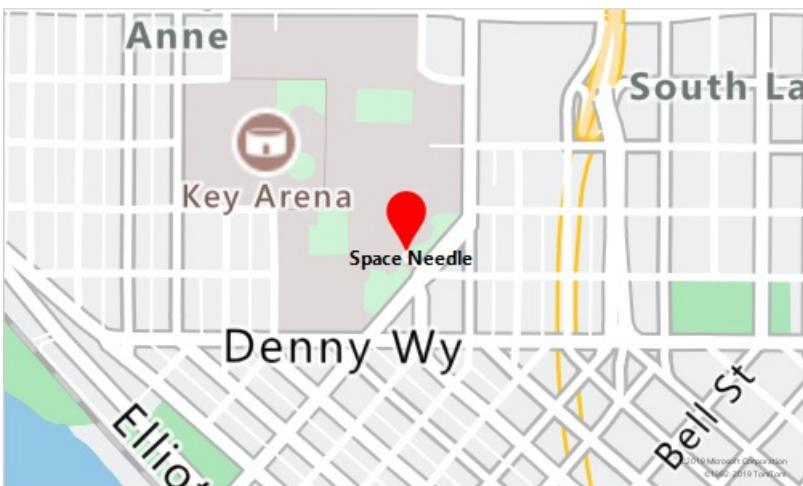
Add pin styles with the `optionNameValue` format. Separate multiple styles with the pipe (|) characters. For example: `iconType|optionName1Value1|optionName2Value2`. The option names and values aren't separated. Use the following style option names to style markers:

- `a1` – Specifies the opacity (alpha) of the marker. Choose a number between 0 and 1.
- `an` – Specifies the pin anchor. Specify X and y pixel values in the "x y" format.
- `co` – The color of the pin. Specify a 24-bit hex color: `000000` to `FFFFFF`.
- `la` – Specifies the label anchor. Specify X and y pixel values in the "x y" format.
- `lc` – The color of the label. Specify a 24-bit hex color: `000000` to `FFFFFF`.
- `ls` – The size of the label in pixels. Choose a number greater than 0.
- `ro` – A value in degrees to rotate the icon. Choose a number between -360 and 360.
- `sc` – A scale value for the pin icon. Choose a number greater than 0.

Specify label values for each pin location. This approach is more efficient than applying a single label value to all markers in the list of locations. The label value can be a string of multiple characters. Wrap the string with single quotes to ensure that it isn't mistaken as a style or location value.

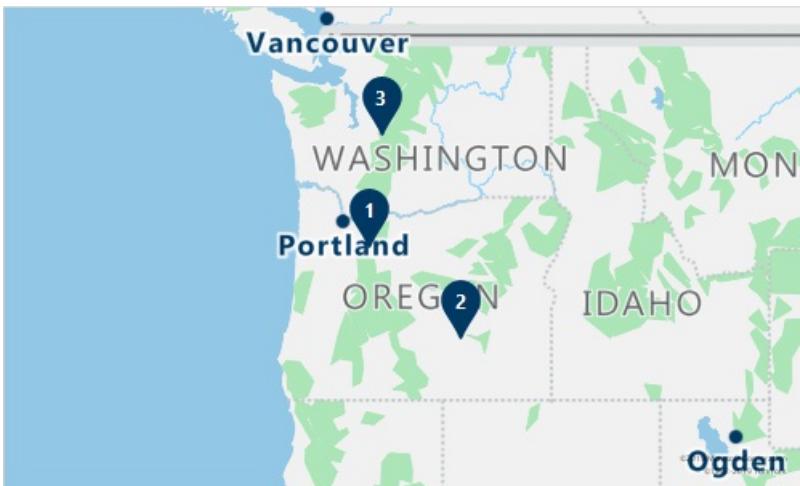
Let's add a red (`FF0000`) default icon, with the label "Space Needle", positioned below (15 50). The icon is at longitude: -122.349300, latitude: 47.620180:

```
&pins=default|coFF0000|la15 50||'Space Needle' -122.349300 47.620180
```



Add three pins with the label values '1', '2', and '3':

```
&pins=default||'1'-122 45||'2'-119.5 43.2||'3'-121.67 47.12
```



Path URL parameter format comparison

Before: Google Maps

Add lines and polygon to a static map image using the `path` parameter in the URL. The `path` parameter takes in a style and a list of locations to be rendered on the map, as shown below:

```
&path=pathStyles|pathLocation1|pathLocation2|...
```

Use additional styles by adding additional `path` parameters to the URL with a different style and set of locations.

Path locations are specified with the `latitude1,longitude1|latitude2,longitude2|...` format. Paths can be encoded or contain addresses for points.

Add path styles with the `optionName:value` format, separate multiple styles by the pipe (|) characters. And, separate option names and values with a colon (:). Like this: `optionName1:value1|optionName2:value2`. The following style option names can be used to style paths in Google Maps:

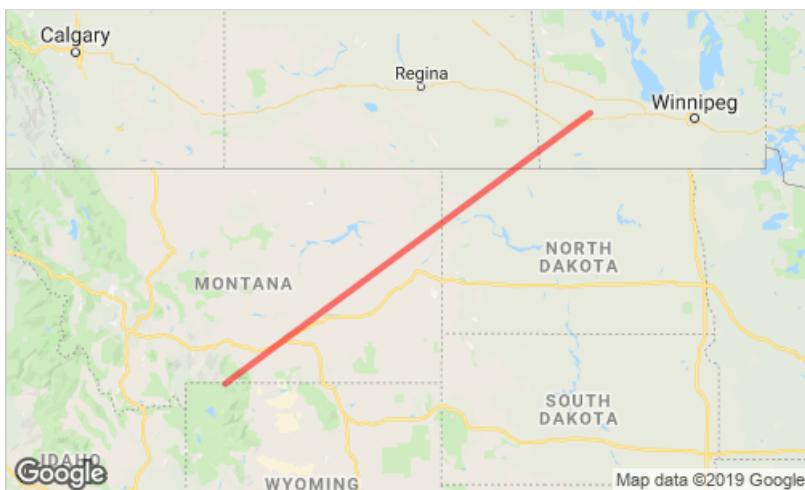
- `color` – The color of the path or polygon outline. Can be a 24-bit hex color (`0xrrggb`), a 32-bit hex color (`0xrrggbbba`) or one of the following values: black, brown, green, purple, yellow, blue, gray, orange, red,

white.

- `fillColor` – The color to fill the path area with (polygon). Can be a 24-bit hex color (`0xrrggbba`), a 32-bit hex color (`0xrrggbbaa`) or one of the following values: black, brown, green, purple, yellow, blue, gray, orange, red, white.
- `geodesic` – Indicates if the path should be a line that follows the curvature of the earth.
- `weight` – The thickness of the path line in pixels.

Add a red line opacity and pixel thickness to the map between the coordinates, in the URL parameter. For the example below, the line has a 50% opacity and a thickness of four pixels. The coordinates are longitude: -110, latitude: 45 and longitude: -100, latitude: 50.

```
&path=color:0xFF000088|weight:4|45,-110|50,-100
```



After: Azure Maps

Add lines and polygons to a static map image by specifying the `path` parameter in the URL. Like Google Maps, specify a style and a list of locations in this parameter. Specify the `path` parameter multiple times to render multiple circles, lines, and polygons with different styles.

```
&path=pathStyles||pathLocation1|pathLocation2|...
```

When it comes to path locations, Azure Maps requires the coordinates to be in "longitude latitude" format. Google Maps uses "latitude,longitude" format. A space, not a comma, separates longitude and latitude in the Azure Maps format. Azure Maps doesn't support encoded paths or addresses for points. Upload larger data sets as a GeoJSON file into the Azure Maps Data Storage API as documented [here](#).

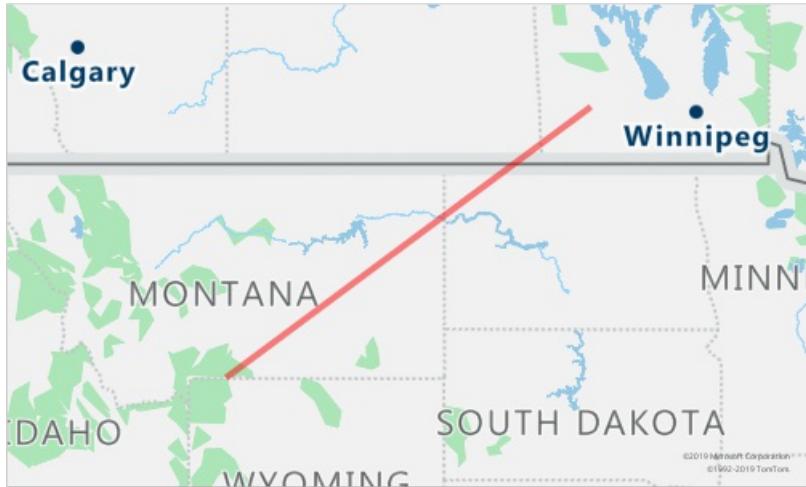
Add path styles with the `optionNameValue` format. Separate multiple styles by pipe (|) characters, like this `optionName1Value1|optionName2Value2`. The option names and values aren't separated. Use the following style option names to style paths in Azure Maps:

- `fa` - The fill color opacity (alpha) used when rendering polygons. Choose a number between 0 and 1.
- `fc` - The fill color used to render the area of a polygon.
- `la` – The line color opacity (alpha) used when rendering lines and the outline of polygons. Choose a number between 0 and 1.
- `lc` – The line color used to render lines and the outline of polygons.
- `lw` – The width of the line in pixels.
- `ra` – Specifies a circles radius in meters.

Add a red line opacity and pixel thickness between the coordinates, in the URL parameter. For the example

below, the line has 50% opacity and a thickness of four pixels. The coordinates have the following values: longitude: -110, latitude: 45 and longitude: -100, latitude: 50.

```
&path=lcFF0000|1a.5|lw4||-110 45|-100 50
```



Calculate a distance matrix

Azure Maps provides the distance matrix API. Use this API to calculate the travel times and the distances between a set of locations, with a distance matrix. It's comparable to the distance matrix API in Google Maps.

- **Route matrix:** Asynchronously calculates travel times and distances for a set of origins and destinations. Supports up to 700 cells per request. That's the number of origins multiplied by the number of destinations. With that constraint in mind, examples of possible matrix dimensions are: 700x1, 50x10, 10x10, 28x25, 10x70.

NOTE

A request to the distance matrix API can only be made using a POST request with the origin and destination information in the body of the request. Additionally, Azure Maps requires all origins and destinations to be coordinates. Addresses will need to be geocoded first.

This table cross-references the Google Maps API parameters with the comparable Azure Maps API parameters.

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
arrival_time	arriveAt
avoid	avoid
departure_time	departAt
destinations	destination – specify in the POST request body as GeoJSON.
key	subscription-key – See also the Authentication with Azure Maps documentation.
language	language – See supported languages documentation.

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
mode	travelMode
origins	origins – specify in the POST request body as GeoJSON.
region	N/A – This feature is geocoding related. Use the countrySet parameter when using the Azure Maps geocoding API.
traffic_model	N/A – Can only specify if traffic data should be used with the traffic parameter.
transit_mode	N/A - Transit-based distance matrices aren't currently supported.
transit_routing_preference	N/A - Transit-based distance matrices aren't currently supported.
units	N/A – Azure Maps only uses the metric system.

TIP

All the advanced routing options available in the Azure Maps routing API are supported in the Azure Maps distance matrix API. Advanced routing options include: truck routing, engine specifications, and so on.

Review the [best practices for routing](#) documentation.

Get a time zone

Azure Maps provides an API for retrieving the time zone of a coordinate. The Azure Maps time zone API is comparable to the time zone API in Google Maps:

- **Time zone by coordinate:** Specify a coordinate and receive the time zone details of the coordinate.

This table cross-references the Google Maps API parameters with the comparable API parameters in Azure Maps.

GOOGLE MAPS API PARAMETER	COMPARABLE AZURE MAPS API PARAMETER
key	subscription-key – See also the Authentication with Azure Maps documentation.
language	language – See supported languages documentation.
location	query
timestamp	timeStamp

In addition to this API, Azure Maps provides a number of time zone APIs. These APIs convert the time based on the names or the IDs of the time zone:

- **Time zone by ID:** Returns current, historical, and future time zone information for the specified IANA time zone ID.

- [Time zone Enum IANA](#): Returns a full list of IANA time zone IDs. Updates to the IANA service are reflected in the system within one day.
- [Time zone Enum Windows](#): Returns a full list of Windows Time Zone IDs.
- [Time zone IANA version](#): Returns the current IANA version number used by Azure Maps.
- [Time zone Windows to IANA](#): Returns a corresponding IANA ID, given a valid Windows Time Zone ID. Multiple IANA IDs may be returned for a single Windows ID.

Client libraries

Azure Maps provides client libraries for the following programming languages:

- JavaScript, TypeScript, Node.js – [documentation](#) | [NPM package](#)

These Open-source client libraries are for other programming languages:

- .NET Standard 2.0 – [GitHub project](#) | [NuGet package](#)

Clean up resources

No resources to be cleaned up.

Next steps

Learn more about Azure Maps REST services:

[Best practices for search](#)

Authentication with Azure Maps

5/27/2021 • 6 minutes to read • [Edit Online](#)

Azure Maps supports two ways to authenticate requests: Shared Key authentication and [Azure Active Directory \(Azure AD\)](#) authentication. This article explains both authentication methods to help guide your implementation of Azure Maps services.

NOTE

To improve secure communication with Azure Maps, we now support Transport Layer Security (TLS) 1.2, and we're retiring support for TLS 1.0 and 1.1. If you currently use TLS 1.x, evaluate your TLS 1.2 readiness and develop a migration plan with the testing described in [Solving the TLS 1.0 Problem](#).

Shared Key authentication

Primary and secondary keys are generated after the Azure Maps account is created. You're encouraged to use the primary key as the subscription key when calling Azure Maps with shared key authentication. Shared Key authentication passes a key generated by an Azure Maps account to an Azure Maps service. For each request to Azure Maps services, add the *subscription key* as a parameter to the URL. The secondary key can be used in scenarios like rolling key changes.

For information about viewing your keys in the Azure portal, see [Manage authentication](#).

NOTE

Primary and Secondary keys should be treated as sensitive data. The shared key is used to authenticate all Azure Maps REST APIs. Users who use a shared key should abstract the API key away, either through environment variables or secure secret storage, where it can be managed centrally.

Azure AD authentication

Azure Subscriptions are provided with an Azure AD tenant to enable fine grained access control. Azure Maps offers authentication for Azure Maps services using Azure AD. Azure AD provides identity-based authentication for users and applications registered in the Azure AD tenant.

Azure Maps accepts **OAuth 2.0** access tokens for Azure AD tenants associated with an Azure subscription that contains an Azure Maps account. Azure Maps also accepts tokens for:

- Azure AD users
- Partner applications that use permissions delegated by users
- Managed identities for Azure resources

Azure Maps generates a *unique identifier (client ID)* for each Azure Maps account. You can request tokens from Azure AD when you combine this client ID with additional parameters.

For more information about how to configure Azure AD and request tokens for Azure Maps, see [Manage authentication in Azure Maps](#).

For general information about authenticating with Azure AD, see [What is authentication?](#).

Managed identities for Azure resources and Azure Maps

Managed identities for Azure resources provide Azure services with an automatically managed application based security principal that can authenticate with Azure AD. With Azure role-based access control (Azure RBAC), the managed identity security principal can be authorized to access Azure Maps services. Some examples of managed identities include: Azure App Service, Azure Functions, and Azure Virtual Machines. For a list of managed identities, see [managed identities for Azure resources](#).

Configuring application Azure AD authentication

Applications will authenticate with the Azure AD tenant using one or more supported scenarios provided by Azure AD. Each Azure AD application scenario represents different requirements based on business needs. Some applications may require user sign-in experiences and other applications may require an application sign-in experience. For more information, see [Authentication flows and application scenarios](#).

After the application receives an access token, the SDK and/or application sends an HTTPS request with the following set of required HTTP headers in addition to other REST API HTTP headers:

HEADER NAME	VALUE
x-ms-client-id	30d7cc....9f55
Authorization	Bearer eyJ0e....HNIVN

NOTE

`x-ms-client-id` is the Azure Maps account-based GUID that appears on the Azure Maps authentication page.

Here's an example of an Azure Maps route request that uses an Azure AD OAuth Bearer token:

```
GET /route/directions/json?api-version=1.0&query=52.50931,13.42936:52.50274,13.43872
Host: atlas.microsoft.com
x-ms-client-id: 30d7cc....9f55
Authorization: Bearer eyJ0e....HNIVN
```

For information about viewing your client ID, see [View authentication details](#).

Authorization with role-based access control

Azure Maps supports access to all principal types for [Azure role-based access control \(Azure RBAC\)](#) including: individual Azure AD users, groups, applications, Azure resources, and Azure Managed identities. Principal types are granted a set of permissions, also known as a role definition. A role definition provides permissions to REST API actions. Applying access to one or more Azure Maps accounts is known as a scope. When applying a principal, role definition, and scope then a role assignment is created.

The next sections discuss concepts and components of Azure Maps integration with Azure RBAC. As part of the process to set up your Azure Maps account, an Azure AD directory is associated to the Azure subscription, which the Azure Maps account resides.

When you configure Azure RBAC, you choose a security principal and apply it to a role assignment. To learn how to add role assignments on the Azure portal, see [Assign Azure roles](#).

Picking a role definition

The following role definition types exist to support application scenarios.

AZURE ROLE DEFINITION	DESCRIPTION
Azure Maps Data Reader	Provides access to immutable Azure Maps REST APIs.
Azure Maps Data Contributor	Provides access to mutable Azure Maps REST APIs. Mutability is defined by the actions: write and delete.
Custom Role Definition	Create a crafted role to enable flexible restricted access to Azure Maps REST APIs.

Some Azure Maps services may require elevated privileges to perform write or delete actions on Azure Maps REST APIs. Azure Maps Data Contributor role is required for services which provide write or delete actions. The following table describes which services Azure Maps Data Contributor is applicable for when using write or delete actions on the given service. If only read actions are used on the service, then Azure Maps Data Reader can be used instead of Azure Maps Data Contributor.

AZURE MAPS SERVICE	AZURE MAPS ROLE DEFINITION
Data	Azure Maps Data Contributor
Creator	Azure Maps Data Contributor
Spatial	Azure Maps Data Contributor

For information about viewing your Azure RBAC settings, see [How to configure Azure RBAC for Azure Maps](#).

Custom role definitions

One aspect of application security is to apply the principle of least privilege. This principle implies that the security principal should only be allowed the access which is required, and have no additional access. Creating custom role definitions can support use cases which require further granularity to access control. To create a custom role definition, you can select specific data actions to include or exclude for the definition.

The custom role definition can then be used in a role assignment for any security principal. To learn more about Azure custom role definitions, see [Azure custom roles](#).

Here are some example scenarios where custom roles can improve application security.

SCENARIO	CUSTOM ROLE DATA ACTION(S)
A public facing or interactive sign-in web page with base map tiles and no other REST APIs.	<code>Microsoft.Maps/accounts/services/render/read</code>
An application which only requires reverse geocoding and no other REST APIs.	<code>Microsoft.Maps/accounts/services/search/read</code>
A role for a security principal which requests reading of Azure Maps Creator based map data and base map tile REST APIs.	<code>Microsoft.Maps/accounts/services/data/read</code> , <code>Microsoft.Maps/accounts/services/render/read</code>
A role for a security principal which requires reading, writing, and deleting of Creator based map data. This can be defined as a map data editor role but does not allow access to other REST APIs like base map tiles.	<code>Microsoft.Maps/accounts/services/data/read</code> , <code>Microsoft.Maps/accounts/services/data/write</code> , <code>Microsoft.Maps/accounts/services/data/delete</code>

Understanding scope

When creating a role assignment, it is defined within the Azure resource hierarchy. At the top of the hierarchy is a [management group](#) and the lowest is an Azure resource, like an Azure Maps account. Assigning a role assignment to a resource group can enable access to multiple Azure Maps accounts or resources in the group.

TIP

Microsoft's general recommendation is to assign access to the Azure Maps account scope because it prevents **unintended access to other Azure Maps accounts** existing in the same Azure subscription.

Next steps

To learn more about Azure RBAC, see

[Azure role-based access control](#)

To learn more about authenticating an application with Azure AD and Azure Maps, see

[Manage authentication in Azure Maps](#)

To learn more about authenticating the Azure Maps Map Control with Azure AD, see

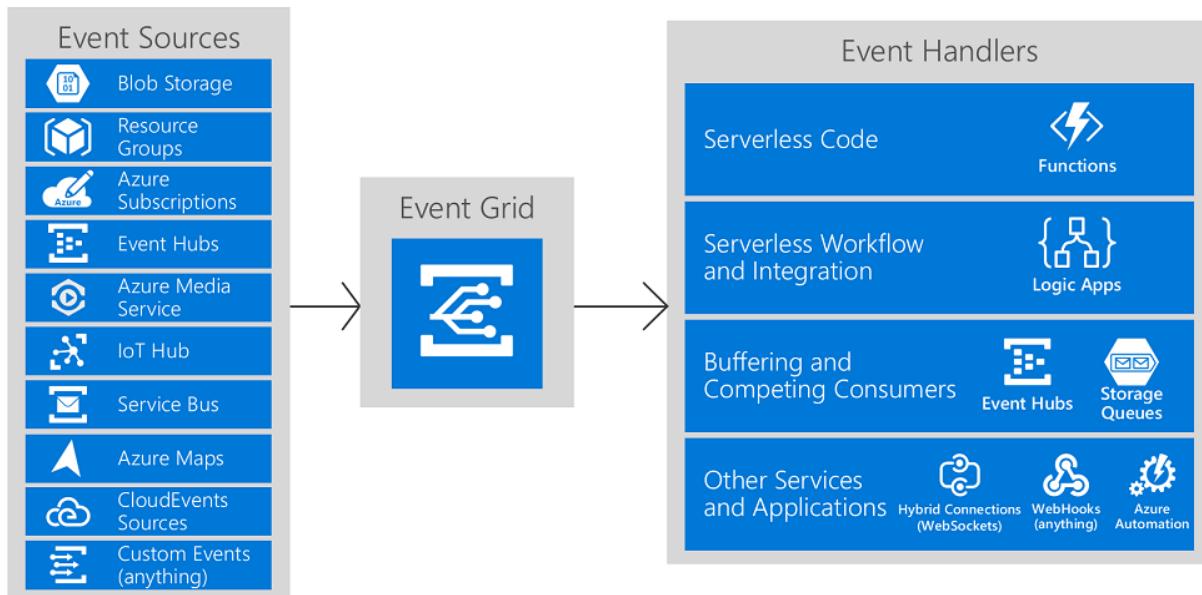
[Use the Azure Maps Map Control](#)

React to Azure Maps events by using Event Grid

11/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Maps integrates with Azure Event Grid, so that users can send event notifications to other services and trigger downstream processes. The purpose of this article is to help you configure your business applications to listen to Azure Maps events. This allows users to react to critical events in a reliable, scalable, and secure manner. For example, users can build an application to update a database, create a ticket, and deliver an email notification, every time a device enters a geofence.

Azure Event Grid is a fully managed event routing service, which uses a publish-subscribe model. Event Grid has built-in support for Azure services like [Azure Functions](#) and [Azure Logic Apps](#). It can deliver event alerts to non-Azure services using webhooks. For a complete list of the event handlers that Event Grid supports, see [An introduction to Azure Event Grid](#).



Azure Maps events types

Event grid uses [event subscriptions](#) to route event messages to subscribers. An Azure Maps account emits the following event types:

EVENT TYPE	DESCRIPTION
MicrosoftMapsGeofenceEntered	Raised when received coordinates have moved from outside of a given geofence to within
MicrosoftMapsGeofenceExited	Raised when received coordinates have moved from within a given geofence to outside
MicrosoftMapsGeofenceResult	Raised every time a geofencing query returns a result, regardless of the state

Event schema

The following example shows the schema for GeofenceResult:

```
{  
  "id": "451675de-a67d-4929-876c-5c2bf0b2c000",  
  
  "topic": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.Maps/accounts/{accountName}",  
  "subject": "/spatial/geofence/udid/{udid}/id/{eventId}",  
  "data": {  
    "geometries": [  
      {  
        "deviceId": "device_1",  
        "udId": "1a13b444-4acf-32ab-ce4e-9ca4af20b169",  
        "geometryId": "1",  
        "distance": 999.0,  
        "nearestLat": 47.609833,  
        "nearestLon": -122.148274  
      }  
    ],  
    "expiredGeofenceGeometryId": [  
    ],  
    "invalidPeriodGeofenceGeometryId": [  
    ]  
  },  
  "eventType": "Microsoft.Maps.GeofenceResult",  
  "eventTime": "2018-11-08T00:52:08.0954283Z",  
  "metadataVersion": "1",  
  "dataVersion": "1.0"  
}
```

Tips for consuming events

Applications that handle Azure Maps geofence events should follow a few recommended practices:

- Configure multiple subscriptions to route events to the same event handler. It's important not to assume that events are from a particular source. Always check the message topic to ensure that the message came from the source that you expect.
- Use the `x-correlation-id` field in the response header to understand if your information about objects is up to date. Messages can arrive out of order or after a delay.
- When a GET or a POST request in the Geofence API is called with the mode parameter set to `EnterAndExit`, then an Enter or Exit event is generated for each geometry in the geofence for which the status has changed from the previous Geofence API call.

Next steps

To learn more about how to use geofencing to control operations at a construction site, see:

[Set up a geofence by using Azure Maps](#)

Choose the right pricing tier in Azure Maps

6/1/2021 • 2 minutes to read • [Edit Online](#)

Azure Maps now offers two pricing tiers: Gen 1 and Gen 2. The Gen 2 new pricing tier contains all Azure Maps capabilities without any QPS (Queries Per Second) restriction and allows you to achieve cost savings as Azure Maps transactions increases. The purpose of this article is to help you choose the right pricing tier for your needs.

Pricing tier targeted customers

See the **pricing tier targeted customers** table below for a better understanding of Gen 1 and Gen 2 pricing tiers. For more information, see [Azure Maps pricing](#). If you're a current Azure Maps customer, you can learn how to change from Gen 1 to Gen 2 pricing [here](#).

PRICING TIER	SKU	TARGETED CUSTOMERS
Gen 1	S0	The S0 pricing tier works for applications in all stages of production: from proof-of-concept development and early stage testing to application production and deployment. However, this tier is designed for small-scale development, or customers with low concurrent users, or both.
	S1	The S1 pricing tier is for customers with large-scale enterprise applications, mission-critical applications, or high volumes of concurrent users. It's also for those customers who require advanced geospatial services.
Gen 2	Maps/Location Insights	Gen 2 pricing is for new and current Azure Maps customers. Gen 2 comes with a free monthly tier of transactions to be used to test and build on Azure maps. Maps and Location Insights SKU's contain all of Azure Maps capabilities. Additionally, there's no QPS (Queries Per Second) restrictions, which for most services, achieves cost savings as Azure Maps transactions increase.

Next steps

Learn more about how to view and change pricing tiers:

[Manage a pricing tier](#)

Creator service geographic scope

6/1/2021 • 2 minutes to read • [Edit Online](#)

Azure Maps Creator is a geographically scoped service. Creator offers a resource provider API that, given an Azure region, creates an instance of Creator data deployed at the geographical level. The mapping from an Azure region to geography happens behind the scenes as described in the table below. For more details on Azure regions and geographies, see [Azure geographies](#).

Data locations

For disaster recovery and high availability, Microsoft may replicate customer data to other regions only within the same geographic area. For example, data in West Europe may be replicated to North Europe, but not to the United States. Regardless, no matter which geography the customer selected, Microsoft doesn't control or limit the locations from which the customers, or their end users, may access customer data via Azure Maps API.

Geographic and regional mapping

The following table describes the mapping between geography and supported Azure regions, and the respective geographic API endpoint. For example, if a Creator account is provisioned in the West US 2 region that falls within the United States geography, all API calls to the Conversion service must be made to

`us.atlas.microsoft.com/conversion/convert` .

AZURE GEOGRAPHIC AREAS (GEOS)	AZURE DATACENTERS (REGIONS)	API GEOGRAPHIC ENDPOINT
Europe	West Europe, North Europe	eu.atlas.microsoft.com
United States	West US 2, East US 2	us.atlas.microsoft.com

Creator for indoor maps

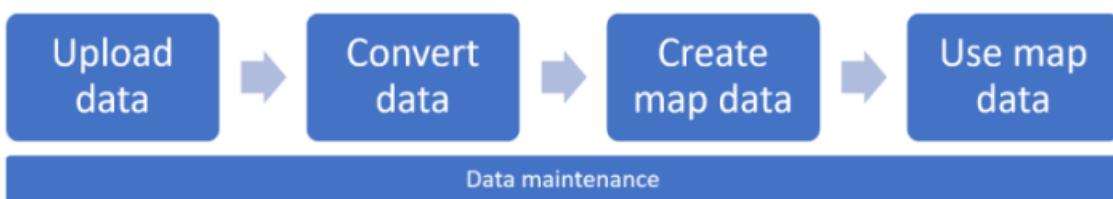
6/9/2021 • 8 minutes to read • [Edit Online](#)

This article introduces concepts and tools that apply to Azure Maps Creator. We recommend that you read this article before you begin to use the Azure Maps Creator API and SDK.

You can use Creator to develop applications with map features that are based on indoor map data. This article describes the process of uploading, converting, creating, and using your map data. Typically, the workflow is completed by two different personas with distinct areas of expertise and responsibility:

- Map maker: responsible for curating and preparing the map data.
- Creator map data user: leverages customer map data in applications.

The following diagram illustrates the entire workflow.



Create Azure Maps Creator

To use Creator services, Azure Maps Creator must be created in an Azure Maps account. For information about how to create Azure Maps Creator in Azure Maps, see [Manage Azure Maps Creator](#).

Creator authentication

Creator inherits Azure Maps Access Control (IAM) settings. All API calls for data access must be sent with authentication and authorization rules.

Creator usage data is incorporated in your Azure Maps usage charts and activity log. For more information, see [Manage authentication in Azure Maps](#).

IMPORTANT

We recommend using:

- Azure Active Directory (Azure AD) in all solutions that are built with an Azure Maps account using Creator services. For more information about Azure AD, see [Azure AD authentication](#).
- Role-based access control settings. Using these settings, map makers can act as the Azure Maps Data Contributor role, and Creator map data users can act as the Azure Maps Data Reader role. For more information, see [Authorization with role-based access control](#).

Creator data item types

Creator services create, store, and use various data types that are defined and discussed in the following sections. A creator data item can be of the following types:

- Converted data

- Dataset
- Tileset
- Feature stateset

Upload a Drawing package

Creator collects indoor map data by converting an uploaded Drawing package. The Drawing package represents a constructed or remodeled facility. For information about Drawing package requirements, see [Drawing package requirements](#).

Use the [Azure Maps Data Upload API](#) to upload a Drawing package. After the Drawing packing is uploaded, the Data Upload API returns a user data identifier (`udid`). The `udid` can then be used to convert the uploaded package into indoor map data.

Convert a Drawing package

The [Azure Maps Conversion service](#) converts an uploaded Drawing package into indoor map data. The Conversion service also validates the package. Validation issues are classified into two types:

- Errors: If any errors are detected, the conversion process fails. When an error occurs, the Conversion service provides a link to the [Azure Maps Drawing Error Visualizer](#) stand-alone web application. You can use the Drawing Error Visualizer to inspect [Drawing package warnings and errors](#) that occurred during the conversion process. After you fix the errors, you can attempt to upload and convert the package.
- Warnings: If any warnings are detected, the conversion succeeds. However, we recommend that you review and resolve all warnings. A warning means that part of the conversion was ignored or automatically fixed. Failing to resolve the warnings could result in errors in later processes. For more information, see [Drawing package warnings and errors](#).

Create indoor map data

Azure Maps Creator provides the following services that support map creation:

- [Dataset service](#).
- [Tileset service](#). Use the Tileset service to create a vector-based representation of a dataset. Applications can use a tileset to present a visual tile-based view of the dataset.
- [Feature State service](#). Use the Feature State service to support dynamic map styling. Applications can use dynamic map styling to reflect real-time events on spaces provided by the IoT system.

Datasets

A dataset is a collection of indoor map features. The indoor map features represent facilities that are defined in a converted Drawing package. After you create a dataset with the [Dataset service](#), you can create any number of [tilesets](#) or [feature statesets](#).

At any time, developers can use the [Dataset service](#) to add or remove facilities to an existing dataset. For more information about how to update an existing dataset using the API, see the append options in [Dataset service](#). For an example of how to update a dataset, see [Data maintenance](#).

Tilesets

A tileset is a collection of vector data that represents a set of uniform grid tiles. Developers can use the [Tileset service](#) to create tilesets from a dataset.

To reflect different content stages, you can create multiple tilesets from the same dataset. For example, you can make one tileset with furniture and equipment, and another tileset without furniture and equipment. You might choose to generate one tileset with the most recent data updates, and another tileset without the most recent

data updates.

In addition to the vector data, the tileset provides metadata for map rendering optimization. For example, tileset metadata contains a minimum and maximum zoom level for the tileset. The metadata also provides a bounding box that defines the geographic extent of the tileset. An application can use a bounding box to programmatically set the correct center point. For more information about tileset metadata, see [Tileset List API](#).

After a tileset is created, it can be retrieved by the [Render V2 service](#).

If a tileset becomes outdated and is no longer useful, you can delete the tileset. For information about how to delete tilesets, see [Data maintenance](#).

NOTE

A tileset is independent of the dataset from which it was created. If you create tilesets from a dataset, and then subsequently update that dataset, the tilesets isn't updated.

To reflect changes in a dataset, you must create new tilesets. Similarly, if you delete a tileset, the dataset isn't affected.

Feature statesets

Feature statesets are collections of dynamic properties (*states*) that are assigned to dataset features, such as rooms or equipment. An example of a *state* can be temperature or occupancy. Each *state* is a key/value pair that contains the name of the property, the value, and the timestamp of the last update.

You can use the [Feature State service](#) to create and manage a feature stateset for a dataset. The stateset is defined by one or more *states*. Each feature, such as a room, can have one *state* attached to it.

The value of each *state* in a stateset can be updated or retrieved by IoT devices or other applications. For example, using the [Feature State Update API](#), devices measuring space occupancy can systematically post the state change of a room.

An application can use a feature stateset to dynamically render features in a facility according to their current state and respective map style. For more information about using feature statesets to style features in a rendering map, see [Indoor Maps module](#).

NOTE

Like tilesets, changing a dataset doesn't affect the existing feature stateset, and deleting a feature stateset doesn't affect the dataset to which it's attached.

Using indoor maps

Render V2-Get Map Tile API

The Azure Maps [Render V2-Get Map Tile API](#) has been extended to support Creator tilesets.

Applications can use the Render V2-Get Map Tile API to request tilesets. The tilesets can then be integrated into a map control or SDK. For an example of a map control that uses the Render V2 service, see [Indoor Maps Module](#).

Web Feature Service API

You can use the [Web Feature Service \(WFS\) API](#) to query datasets. WFS follows the [Open Geospatial Consortium API Features](#). You can use the WFS API to query features within the dataset itself. For example, you can use WFS to find all mid-size meeting rooms of a specific facility and floor level.

Alias API

Creator services such as Conversion, Dataset, Tileset, and Feature State return an identifier for each resource

that's created from the APIs. The [Alias API](#) allows you to assign an alias to reference a resource identifier.

Indoor Maps module

The [Azure Maps Web SDK](#) includes the Indoor Maps module. This module offers extended functionalities to the Azure Maps *Map Control* library. The Indoor Maps module renders indoor maps created in Creator. It integrates widgets, such as *floor picker*, that help users to visualize the different floors.

You can use the Indoor Maps module to create web applications that integrate indoor map data with other [Azure Maps services](#). The most common application setups include adding knowledge from other maps - such as road, imagery, weather, and transit - to indoor maps.

The Indoor Maps module also supports dynamic map styling. For a step-by-step walkthrough to implement feature stateset dynamic styling in an application, see [Use the Indoor Map module](#).

Azure Maps integration

As you begin to develop solutions for indoor maps, you can discover ways to integrate existing Azure Maps capabilities. For example, you can implement asset tracking or safety scenarios by using the [Azure Maps Geofence API](#) with Creator indoor maps. For example, you can use the Geofence API to determine whether a worker enters or leaves specific indoor areas. For more information about how to connect Azure Maps with IoT telemetry, see [this IoT spatial analytics tutorial](#).

Data maintenance

You can use the Azure Maps Creator List, Update, and Delete API to list, update, and delete your datasets, tilesets, and feature statesets.

NOTE

When you review a list of items to determine whether to delete them, consider the impact of that deletion on all dependent API or applications. For example, if you delete a tileset that's being used by an application by means of the [Render V2-Get Map Tile API](#), the application fails to render that tileset.

Example: Updating a dataset

The following example shows how to update a dataset, create a new tileset, and delete an old tileset:

1. Follow steps in the [Upload a Drawing package](#) and [Convert a Drawing package](#) sections to upload and convert the new Drawing package.
2. Use the [Dataset Create API](#) to append the converted data to the existing dataset.
3. Use the [Tileset Create API](#) to generate a new tileset out of the updated dataset.
4. Save the new `tilesetId` for the next step.
5. To enable the visualization of the updated campus dataset, update the tileset identifier in your application. If the old tileset is no longer used, you can delete it.

Next steps

[Tutorial: Creating a Creator indoor map](#)

Drawing conversion errors and warnings

6/1/2021 • 10 minutes to read • [Edit Online](#)

The [Azure Maps Conversion service](#) lets you convert uploaded Drawing packages into map data. Drawing packages must adhere to the [Drawing package requirements](#). If one or more requirements aren't met, then the Conversion service will return errors or warnings. This article lists the conversion error and warning codes, with recommendations on how to resolve them. It also provides some examples of drawings that can cause the Conversion service to return these codes.

The Conversion service will succeed if there are any conversion warnings. However, it's recommended that you review and resolve all warnings. A warning means part of the conversion was ignored or automatically fixed. Failing to resolve the warnings could result in errors in latter processes.

General Warnings

geometryWarning

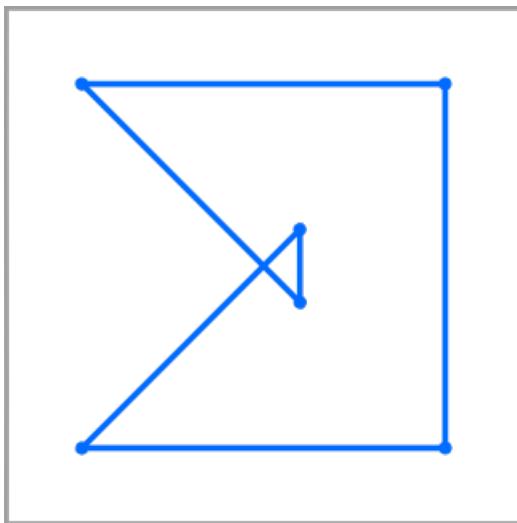
Description for geometryWarning

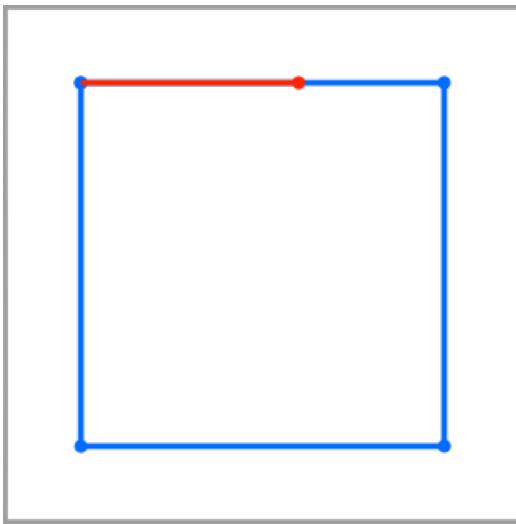
A **geometryWarning** occurs when the drawing contains an invalid entity. An invalid entity is an entity that doesn't conform to geometric constraints. Examples of an invalid entity are a self-intersecting polygon or a non-closed PolyLine in a layer that only supports closed geometry.

The Conversion service is unable to create a map feature from an invalid entity and instead ignores it.

Examples for geometryWarning

- The two images below show examples of self-intersecting polygons.





- Below is an image that shows a non-closed PolyLine. Assume that the layer only supports closed geometry.



How to fix geometryWarning

Inspect the **geometryWarning** for each entity to verify that it follows geometric constraints.

unexpectedGeometryInLayer

Description for unexpectedGeometryInLayer

An **unexpectedGeometryInLayer** warning occurs when the drawing contains geometry that is incompatible with the expected geometry type for a given layer. When the Conversion service returns an **unexpectedGeometryInLayer** warning, it will ignore that geometry.

Example for unexpectedGeometryInLayer

The image below shows a non-closed PolyLine. Assume that the layer only supports closed geometry.



How to fix unexpectedGeometryInLayer

Inspect each `unexpectedGeometryInLayer` warning and move the incompatible geometry to a compatible layer. If it isn't compatible with any of the other layers, it should be removed.

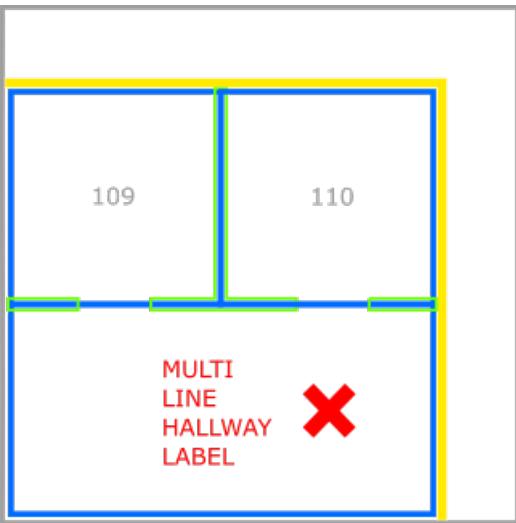
unsupportedFeatureRepresentation

Description for unsupportedFeatureRepresentation

The `unsupportedFeatureRepresentation` warning occurs when the drawing contains an unsupported entity type.

Example for unsupportedFeatureRepresentation

The image below shows an unsupported entity type as a multi-line text object on a label layer.



How to fix unsupportedFeatureRepresentation

Ensure that your DWG files contain only the supported entity types. Supported types are listed under the [Drawing files requirements section in the Drawing package requirements article](#).

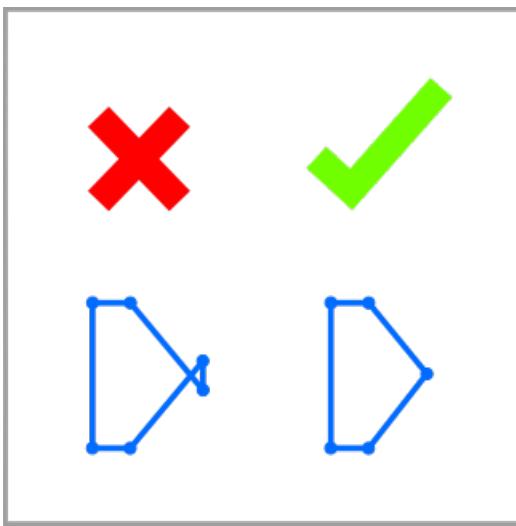
automaticRepairPerformed

Description for automaticRepairPerformed

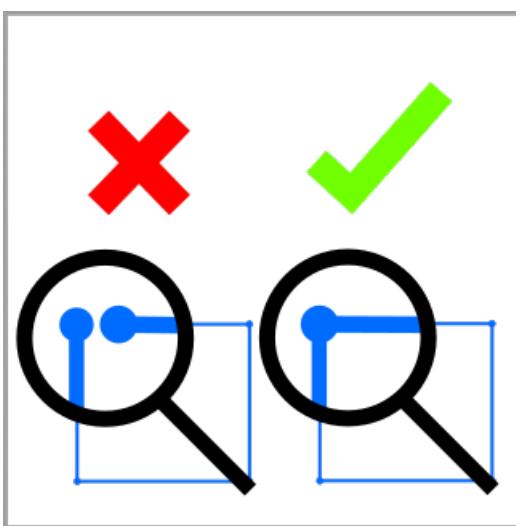
The `automaticRepairPerformed` warning occurs when the Conversion service automatically repairs invalid geometry.

Examples for automaticRepairPerformed

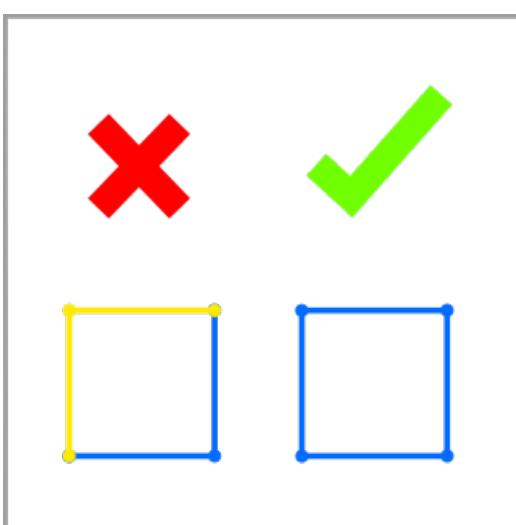
- The following image shows how the Conversion service repaired a self-intersecting polygon into valid geometry.



- The image below shows how the Conversion service snapped the first and last vertex of a non-closed PolyLine to create a closed PolyLine, where the first and last vertex were less than 1 mm apart.



- The image below shows how, in a layer that supports only closed PolyLines, the Conversion service repaired multiple non-closed PolyLines. To avoid discarding the non-closed PolyLines, the service combined them into a single closed PolyLine.



How to fix automaticRepairPerformed

To fix an **automaticRepairPerformed** warning, take the following actions:

- Inspect each warning's geometry and the specific warning text.
- Determine if the automated repair is correct.

- If the repair is correct, continue. Otherwise, go to the design file and resolve the warning manually.

TIP

To suppress a warning in the future, make changes to the original drawing so that the original drawing matches the repaired drawing.

Manifest warnings

redundantAttribution

Description for redundantAttribution

The **redundantAttribution** warning occurs when the manifest contains redundant or conflicting object properties.

Examples for redundantAttribution

- The JSON snippet below contains two or more `unitProperties` objects with the same `name`.

```
"unitProperties": [  
    {  
        "unitName": "L1-100",  
        "categoryName": "room.office"  
    },  
    {  
        "unitName": "L1-101",  
        "categoryName": "room.office"  
    },  
    {  
        "unitName": "L1-101",  
        "categoryName": "room.office"  
    }  
]
```

- In the JSON snippet below, two or more `zoneProperties` objects have the same `name`.

```
"zoneProperties": [  
    {  
        "zoneName": "Assembly Area 1",  
        "categoryName": "zone.assembly"  
    },  
    {  
        "zoneName": "Assembly Area 2",  
        "categoryName": "zone.assembly"  
    },  
    {  
        "zoneName": "Assembly Area 2",  
        "categoryName": "zone.assembly"  
    }  
]
```

How to fix redundantAttribution

To fix a ***redundantAttribution** warning, remove redundant or conflicting object properties.

manifestWarning

Description for manifestWarning

A **manifestWarning** occurs when the manifest contains unitProperties or zoneProperties objects that are unused during conversion.

Examples for manifestWarning

- The manifest contains a `unitProperties` object with a `unitName` that has no matching label in a

`unitLabel` layer.

- The manifest contains a `zoneProperties` object with a `zoneName` that has no matching label in a `zoneLabel` layer.

How to fix manifestWarning

To fix a `manifestWarning`, remove the unused `unitProperties` or `zoneProperties` object from the manifest, or add a unit/zone label to the drawing so that the properties object is used during conversion.

Wall warnings

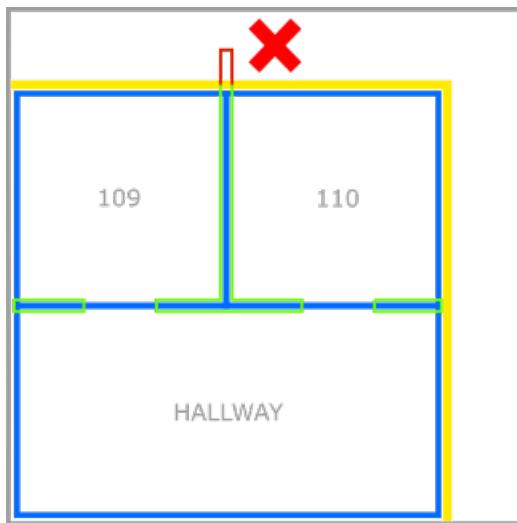
wallOutsideLevel

Description for wallOutsideLevel

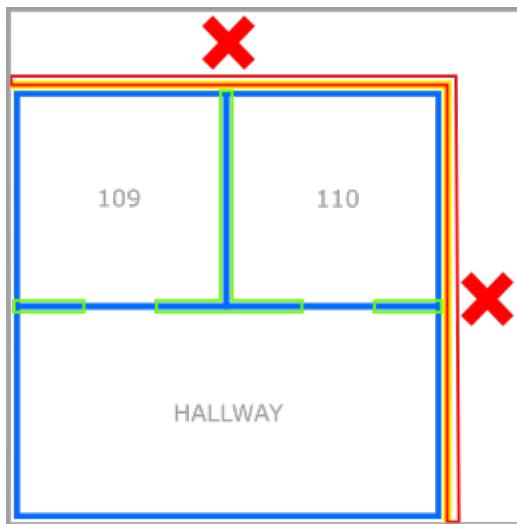
The `wallOutsideLevel` warning occurs when the drawing contains a Wall geometry outside the bounds of a level outline.

Example for wallOutsideLevel

- The image below shows an interior wall, in red, outside the yellow level boundary.



- The following image shows an exterior wall, in red, outside the yellow level boundary.



How to fix wallOutsideLevel

To fix a `wallOutsideLevel` warning, expand the level geometry to include all walls. Or, modify wall boundaries to fit inside the level boundary.

Unit warnings

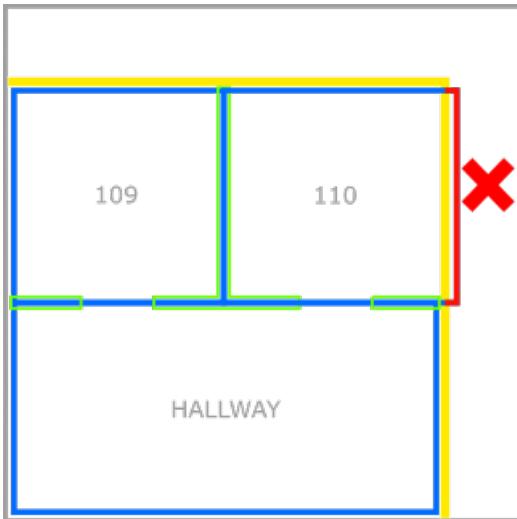
unitOutsideLevel

Description for unitOutsideLevel

A **unitOutsideLevel** warning occurs when the drawing contains unit geometry outside the bounds of the level outline.

Example for unitOutsideLevel

In the following image, unit geometry, in red, exceeds the bounds of the yellow level boundary.



How to fix unitOutsideLevel

To fix a **unitOutsideLevel** warning, expand the level boundary to include all units. Or, modify unit geometry to fit inside the level boundary.

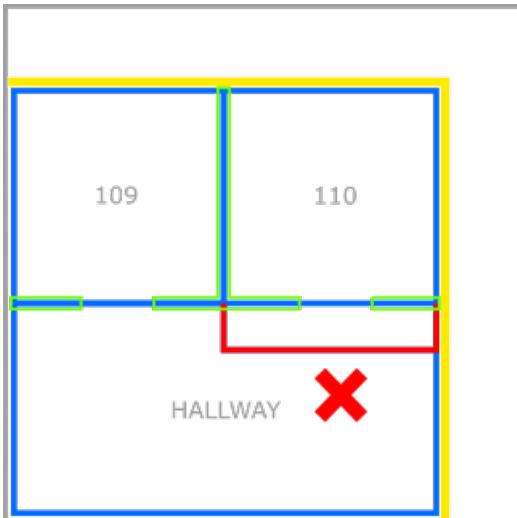
partiallyOverlappingUnit

Description for partiallyOverlappingUnit

A **partiallyOverlappingUnit** warning occurs when the drawing contains a unit geometry partially overlapping on another unit geometry. The Conversion service discards overlapping units.

Example scenarios partiallyOverlappingUnit

In the following image, the overlapping unit is highlighted in red. **UNIT110** and **HALLWAY** are discarded.



How to fix partiallyOverlappingUnit

To fix a **partiallyOverlappingUnit** warning, redraw each partially overlapping unit so that it doesn't overlap any other units.

Door warnings

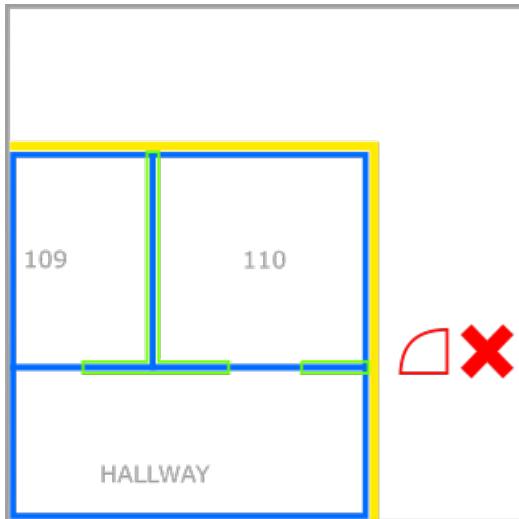
doorOutsideLevel

Description for doorOutsideLevel

A **doorOutsideLevel** warning occurs when the drawing contains a door geometry outside the bounds of the level geometry.

Example for doorOutsideLevel

In the following image, the door geometry, highlighted in red, overlaps the yellow level boundary.



How to fix doorOutsideLevel

To fix a **doorOutsideLevel** warning, redraw your door geometry so that it's inside the level boundaries.

Zone warnings

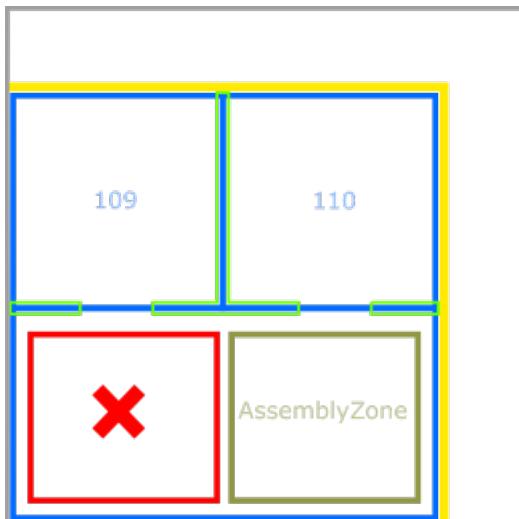
zoneWarning

Description for zoneWarning

The **zoneWarning** occurs when a zone doesn't contain a label. The Conversion service discards a zone that isn't label.I

Example for zoneWarning

The following image shows a zone that doesn't contain a label.



How to fix zoneWarning

To fix a **zoneWarning**, verify that each zone has a single label.

Label Warnings

labelWarning

Description for labelWarning

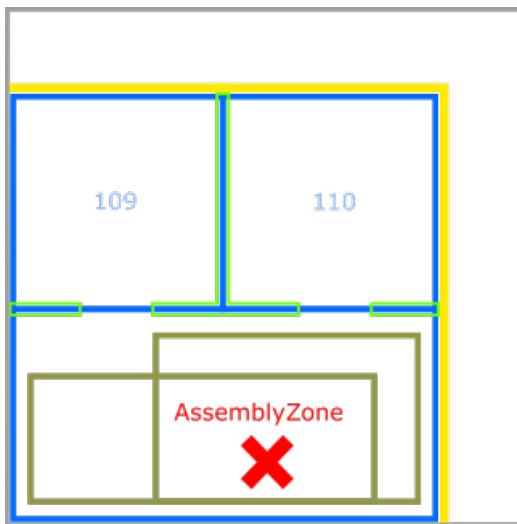
The **labelWarning** occurs when the drawing contains ambiguous or contradictory labels feature.

A **labelWarning** occurs because of one or more of the following reasons:

- A unit label isn't in any units.
- A zone label isn't in any zones.
- A zone label is inside two or more zones.

Example for labelWarning

The following image shows a label that inside two zones.



How to fix labelWarning

To fix a **labelWarning**, ensure that:

- All unit labels are inside units.
- All zone labels are inside zones.
- All zone labels are in one and only one zone.

Drawing Package errors

invalidArchiveFormat

Description for invalidArchiveFormat

An **invalidArchiveFormat** error occurs when the drawing package is in an invalid archive format such as GZIP or 7-Zip. Only the ZIP archive format is supported.

An **invalidArchiveFormat** error will also occur if the ZIP archive is empty.

How to fix invalidArchiveFormat

To fix an **invalidArchiveFormat** error, verify that:

- Your archive file name ends in `.zip`.
- Your ZIP archive contains data.
- You can open your ZIP archive.

invalidUserData

Description for invalidUserData

An **invalidUserData** error occurs when the Conversion service is unable to read a user data object from storage.

Example scenario for invalidUserData

You attempted to upload a Drawing package with an incorrect `udid` parameter.

How to fix invalidUserData

To fix an **invalidUserData** error, verify that:

- You've provided a correct `uid` for the uploaded package.
- Azure Maps Creator has been enabled for the Azure Maps account you used for uploading the Drawing package.
- The API request to the Conversion service contains the subscription key to the Azure Maps account you used for uploading the Drawing package.

dwgError

Description for dwgError

A **dwgError** when the drawing package contains an issue with one or more DWG files in the uploaded ZIP archive.

The **dwgError** occurs when the drawing package contains a DWG file that can't be opened because it's invalid or corrupt.

- A DWG file isn't a valid AutoCAD DWG file format drawing.
- A DWG file is corrupt.
- A DWG file is listed in the *manifest.json* file, but it's missing from the ZIP archive.

How to fix dwgError

To fix a **dwgError**, inspect your *manifest.json* file confirm that:

- All DWG files in your ZIP archive are valid AutoCAD DWG format drawings, open each one in AutoCAD. Remove or fix all invalid drawings.
- The list of DWG files in the *manifest.json* matches the DWG files in the ZIP archive.

Manifest errors

invalidJsonFormat

Description for invalidJsonFormat

An **invalidJsonFormat** error occurs when the *manifest.json* file can't be read.

The `_manifest.json_file` can't be read because of JSON formatting or syntax errors. To learn more about how JSON format and syntax, see [The JavaScript Object Notation \(JSON\) Data Interchange Format](#)

How to fix invalidJsonFormat

To fix an **invalidJsonFormat** error, use a JSON linter to detect and resolve any JSON errors.

missingRequiredField

Description for missingRequiredField

A **missingRequiredField** error occurs when the *manifest.json* file is missing required data.

How to fix missingRequiredField

To fix a **missingRequiredField** error, verify that the manifest contains all required properties. For a full list of required manifest object, see the [manifest section in the Drawing package requirements](#)

missingManifest

Description for missingManifest

The **missingManifest** error occurs when the *manifest.json* file is missing from the ZIP archive.

The **missingManifest** error occurs because of one or more of the following reasons:

- The *manifest.json* file is misspelled.
- The *manifest.json* is missing.
- The *manifest.json* isn't inside the root directory of the ZIP archive.

How to fix missingManifest

To fix a **missingManifest** error, confirm that the archive has a file named *manifest.json* at the root level of the ZIP archive.

conflict

Description for conflict

The **conflict** error occurs when the *manifest.json* file contains conflicting information.

Example scenario for conflict

The Conversion service will return a **conflict** error when more than one level is defined with the same level ordinal. The following JSON snippet shows two levels defined with the same ordinal.

```
"buildingLevels":  
{  
    "levels": [  
        {  
            "levelName": "Ground",  
            "ordinal": 0,  
            "filename": "./Level_0.dwg"  
        },  
        {  
            "levelName": "Parking",  
            "ordinal": 0,  
            "filename": "./Level_P.dwg"  
        }  
    ]  
}
```

How to fix conflict

To fix a **conflict** error, inspect your *manifest.json* and remove any conflicting information.

invalidGeoreference

Description for invalidGeoreference

The **invalidGeoreference** error occurs when a *manifest.json* file contains an invalid georeference.

The **invalidGeoreference** error occurs because of one or more of the following reasons:

- The user is georeferencing a latitude or longitude value that is out of range.
- The user is georeferencing a rotation value that is out of range.

Example scenario for invalidGeoreference

In the JSON snippet below, the latitude is above the upper limit.

```
"georeference":  
{  
    "lat": 88.0,  
    "lon": -122.132600,  
    "angle": 0  
},
```

How to fix invalidGeoreference

To fix an **invalidGeoreference** error, verify that the georeferenced values are within range.

IMPORTANT

In GeoJSON, the coordinates order is longitude and latitude. If you don't use the correct order, you may accidentally refer a latitude or longitude value that is out of range.

Wall errors

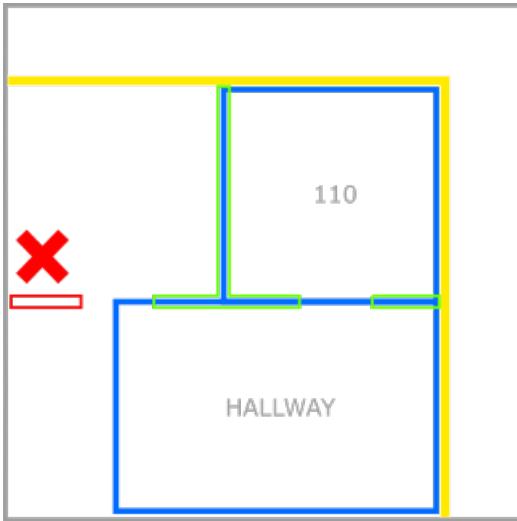
wallError

Description for wallError

The **wallError** occurs when the drawing contains an error while attempting to create a wall feature.

Example scenario for wallError

The following image displays a wall feature that doesn't overlap any units.



How to fix wallError

To fix a **wallError** error, redraw the wall so that it overlaps at least one unit. Or, create a new unit that overlaps the wall.

Vertical Penetration errors

verticalPenetrationError

Description for verticalPenetrationError

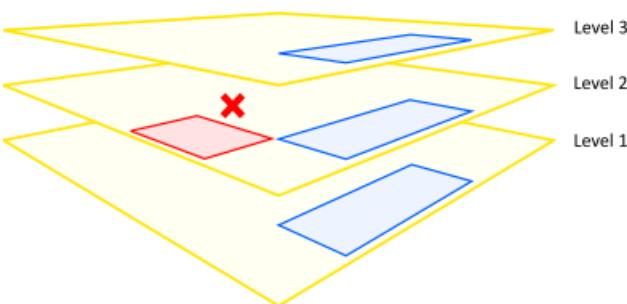
The **verticalPenetrationError** occurs when the drawing contains an ambiguous vertical penetration feature.

The **verticalPenetrationError** occurs because of one or more of the following reasons:

- The drawing contains a vertical penetration area with no overlapping vertical penetration areas on any levels above or below it.
- The drawing package contains a level with two or more vertical penetration features on it that both overlap a single vertical penetration feature on another level directly above or below it.

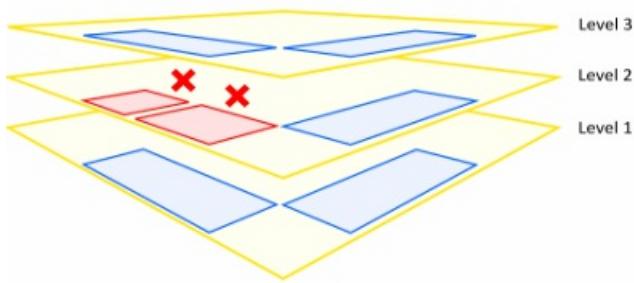
Example scenario for verticalPenetrationError

The image below shows a vertical penetration area with no overlapping vertical penetration areas on levels above or below it.



The following image shows a vertical penetration area that overlaps more than one vertical penetration area on

an adjacent level.



How to fix verticalPenetrationError

To fix a `verticalPenetrationError` error, read about how to use a vertical penetration feature in the [Drawing package requirements](#) article.

Next steps

[How to use Azure Maps Drawing error visualizer](#)

[Drawing Package Guide](#)

[Creator for indoor mapping](#)

Azure Data Encryption at rest

4/27/2021 • 11 minutes to read • [Edit Online](#)

Microsoft Azure includes tools to safeguard data according to your company's security and compliance needs.

This paper focuses on:

- How data is protected at rest across Microsoft Azure
- Discusses the various components taking part in the data protection implementation,
- Reviews pros and cons of the different key management protection approaches.

Encryption at Rest is a common security requirement. In Azure, organizations can encrypt data at rest without the risk or cost of a custom key management solution. Organizations have the option of letting Azure completely manage Encryption at Rest. Additionally, organizations have various options to closely manage encryption or encryption keys.

What is encryption at rest?

Encryption is the secure encoding of data used to protect confidentiality of data. The Encryption at Rest designs in Azure use symmetric encryption to encrypt and decrypt large amounts of data quickly according to a simple conceptual model:

- A symmetric encryption key is used to encrypt data as it is written to storage.
- The same encryption key is used to decrypt that data as it is readied for use in memory.
- Data may be partitioned, and different keys may be used for each partition.
- Keys must be stored in a secure location with identity-based access control and audit policies. Data encryption keys are often encrypted with a key encryption key in Azure Key Vault to further limit access.

In practice, key management and control scenarios, as well as scale and availability assurances, require additional constructs. Microsoft Azure Encryption at Rest concepts and components are described below.

The purpose of encryption at rest

Encryption at rest provides data protection for stored data (at rest). Attacks against data at-rest include attempts to obtain physical access to the hardware on which the data is stored, and then compromise the contained data. In such an attack, a server's hard drive may have been mishandled during maintenance allowing an attacker to remove the hard drive. Later the attacker would put the hard drive into a computer under their control to attempt to access the data.

Encryption at rest is designed to prevent the attacker from accessing the unencrypted data by ensuring the data is encrypted when on disk. If an attacker obtains a hard drive with encrypted data but not the encryption keys, the attacker must defeat the encryption to read the data. This attack is much more complex and resource consuming than accessing unencrypted data on a hard drive. For this reason, encryption at rest is highly recommended and is a high priority requirement for many organizations.

Encryption at rest may also be required by an organization's need for data governance and compliance efforts. Industry and government regulations such as HIPAA, PCI and FedRAMP, lay out specific safeguards regarding data protection and encryption requirements. Encryption at rest is a mandatory measure required for compliance with some of those regulations. For more information on Microsoft's approach to FIPS 140-2 validation, see [Federal Information Processing Standard \(FIPS\) Publication 140-2](#).

In addition to satisfying compliance and regulatory requirements, encryption at rest provides defense-in-depth

protection. Microsoft Azure provides a compliant platform for services, applications, and data. It also provides comprehensive facility and physical security, data access control, and auditing. However, it's important to provide additional "overlapping" security measures in case one of the other security measures fails and encryption at rest provides such a security measure.

Microsoft is committed to encryption at rest options across cloud services and giving customers control of encryption keys and logs of key use. Additionally, Microsoft is working towards encrypting all customer data at rest by default.

Azure Encryption at Rest Components

As described previously, the goal of encryption at rest is that data that is persisted on disk is encrypted with a secret encryption key. To achieve that goal secure key creation, storage, access control, and management of the encryption keys must be provided. Though details may vary, Azure services Encryption at Rest implementations can be described in terms illustrated in the following diagram.



Azure Key Vault

The storage location of the encryption keys and access control to those keys is central to an encryption at rest model. The keys need to be highly secured but manageable by specified users and available to specific services. For Azure services, Azure Key Vault is the recommended key storage solution and provides a common management experience across services. Keys are stored and managed in key vaults, and access to a key vault can be given to users or services. Azure Key Vault supports customer creation of keys or import of customer keys for use in customer-managed encryption key scenarios.

Azure Active Directory

Permissions to use the keys stored in Azure Key Vault, either to manage or to access them for Encryption at Rest encryption and decryption, can be given to Azure Active Directory accounts.

Key Hierarchy

More than one encryption key is used in an encryption at rest implementation. Storing an encryption key in Azure Key Vault ensures secure key access and central management of keys. However, service local access to encryption keys is more efficient for bulk encryption and decryption than interacting with Key Vault for every data operation, allowing for stronger encryption and better performance. Limiting the use of a single encryption key decreases the risk that the key will be compromised and the cost of re-encryption when a key must be replaced. Azure encryptions at rest models use a key hierarchy made up of the following types of keys in order to address all these needs:

- **Data Encryption Key (DEK)** – A symmetric AES256 key used to encrypt a partition or block of data. A single resource may have many partitions and many Data Encryption Keys. Encrypting each block of data with a different key makes crypto analysis attacks more difficult. Access to DEKs is needed by the resource

provider or application instance that is encrypting and decrypting a specific block. When a DEK is replaced with a new key only the data in its associated block must be re-encrypted with the new key.

- **Key Encryption Key (KEK)** – An encryption key used to encrypt the Data Encryption Keys. Use of a Key Encryption Key that never leaves Key Vault allows the data encryption keys themselves to be encrypted and controlled. The entity that has access to the KEK may be different than the entity that requires the DEK. An entity may broker access to the DEK to limit the access of each DEK to a specific partition. Since the KEK is required to decrypt the DEKs, the KEK is effectively a single point by which DEKs can be effectively deleted by deletion of the KEK.

The Data Encryption Keys, encrypted with the Key Encryption Keys are stored separately and only an entity with access to the Key Encryption Key can decrypt these Data Encryption Keys. Different models of key storage are supported. See [data encryption models](#) for more information.

Encryption at rest in Microsoft cloud services

Microsoft Cloud services are used in all three cloud models: IaaS, PaaS, SaaS. Below you have examples of how they fit on each model:

- Software services, referred to as Software as a Server or SaaS, which have applications provided by the cloud such as Microsoft 365.
- Platform services which customers leverage the cloud in their applications, using the cloud for things like storage, analytics, and service bus functionality.
- Infrastructure services, or Infrastructure as a Service (IaaS) in which customer deploys operating systems and applications that are hosted in the cloud and possibly leveraging other cloud services.

Encryption at rest for SaaS customers

Software as a Service (SaaS) customers typically have encryption at rest enabled or available in each service. Microsoft 365 has several options for customers to verify or enable encryption at rest. For information about Microsoft 365 services, see [Encryption in Microsoft 365](#).

Encryption at rest for PaaS customers

Platform as a Service (PaaS) customer's data typically resides in a storage service such as Blob Storage but may also be cached or stored in the application execution environment, such as a virtual machine. To see the encryption at rest options available to you, examine the [Data encryption models: supporting services table](#) for the storage and application platforms that you use.

Encryption at rest for IaaS customers

Infrastructure as a Service (IaaS) customers can have a variety of services and applications in use. IaaS services can enable encryption at rest in their Azure hosted virtual machines and VHDs using Azure Disk Encryption.

Encrypted storage

Like PaaS, IaaS solutions can leverage other Azure services that store data encrypted at rest. In these cases, you can enable the Encryption at Rest support as provided by each consumed Azure service. The [Data encryption models: supporting services table](#) enumerates the major storage, services, and application platforms and the model of Encryption at Rest supported.

Encrypted compute

All Managed Disks, Snapshots, and Images are encrypted using Storage Service Encryption using a service-managed key. A more complete Encryption at Rest solution ensures that the data is never persisted in unencrypted form. While processing the data on a virtual machine, data can be persisted to the Windows page file or Linux swap file, a crash dump, or to an application log. To ensure this data is encrypted at rest, IaaS applications can use Azure Disk Encryption on an Azure IaaS virtual machine (Windows or Linux) and virtual disk.

Custom encryption at rest

It is recommended that whenever possible, IaaS applications leverage Azure Disk Encryption and Encryption at Rest options provided by any consumed Azure services. In some cases, such as irregular encryption requirements or non-Azure based storage, a developer of an IaaS application may need to implement encryption at rest themselves. Developers of IaaS solutions can better integrate with Azure management and customer expectations by leveraging certain Azure components. Specifically, developers should use the Azure Key Vault service to provide secure key storage as well as provide their customers with consistent key management options with that of most Azure platform services. Additionally, custom solutions should use Azure-Managed Service Identities to enable service accounts to access encryption keys. For developer information on Azure Key Vault and Managed Service Identities, see their respective SDKs.

Azure resource providers encryption model support

Microsoft Azure Services each support one or more of the encryption at rest models. For some services, however, one or more of the encryption models may not be applicable. For services that support customer-managed key scenarios, they may support only a subset of the key types that Azure Key Vault supports for key encryption keys. Additionally, services may release support for these scenarios and key types at different schedules. This section describes the encryption at rest support at the time of this writing for each of the major Azure data storage services.

Azure disk encryption

Any customer using Azure Infrastructure as a Service (IaaS) features can achieve encryption at rest for their IaaS VMs and disks through Azure Disk Encryption. For more information on Azure Disk encryption, see the [Azure Disk Encryption documentation](#).

Azure storage

All Azure Storage services (Blob storage, Queue storage, Table storage, and Azure Files) support server-side encryption at rest; some services additionally support customer-managed keys and client-side encryption.

- Server-side: All Azure Storage Services enable server-side encryption by default using service-managed keys, which is transparent to the application. For more information, see [Azure Storage Service Encryption for Data at Rest](#). Azure Blob storage and Azure Files also support RSA 2048-bit customer-managed keys in Azure Key Vault. For more information, see [Storage Service Encryption using customer-managed keys in Azure Key Vault](#).
- Client-side: Azure Blobs, Tables, and Queues support client-side encryption. When using client-side encryption, customers encrypt the data and upload the data as an encrypted blob. Key management is done by the customer. For more information, see [Client-Side Encryption and Azure Key Vault for Microsoft Azure Storage](#).

Azure SQL Database

Azure SQL Database currently supports encryption at rest for Microsoft-managed service side and client-side encryption scenarios.

Support for server encryption is currently provided through the SQL feature called Transparent Data Encryption. Once an Azure SQL Database customer enables TDE key are automatically created and managed for them. Encryption at rest can be enabled at the database and server levels. As of June 2017, [Transparent Data Encryption \(TDE\)](#) is enabled by default on newly created databases. Azure SQL Database supports RSA 2048-bit customer-managed keys in Azure Key Vault. For more information, see [Transparent Data Encryption with Bring Your Own Key support for Azure SQL Database and Data Warehouse](#).

Client-side encryption of Azure SQL Database data is supported through the [Always Encrypted](#) feature. Always Encrypted uses a key that created and stored by the client. Customers can store the master key in a Windows certificate store, Azure Key Vault, or a local Hardware Security Module. Using SQL Server Management Studio, SQL users choose what key they'd like to use to encrypt which column.

Conclusion

Protection of customer data stored within Azure Services is of paramount importance to Microsoft. All Azure hosted services are committed to providing Encryption at Rest options. Azure services support either service-managed keys, customer-managed keys, or client-side encryption. Azure services are broadly enhancing Encryption at Rest availability and new options are planned for preview and general availability in the upcoming months.

Next steps

- See [data encryption models](#) to learn more about service-managed keys and customer-managed keys.
- Learn how Azure uses [double encryption](#) to mitigate threats that come with encrypting data.
- Learn what Microsoft does to ensure [platform integrity and security](#) of hosts traversing the hardware and firmware build-out, integration, operationalization, and repair pipelines.

Geographic coverage information

11/2/2020 • 2 minutes to read • [Edit Online](#)

The following links provide detail coverage information for each of the services in Azure Maps.

- [Geocoding and search coverage](#)
- [Traffic coverage](#)
- [Render coverage](#)
- [Routing coverage](#)
- [Mobility coverage](#)
- [Weather coverage](#)

Next steps

Learn about supported regions, languages, and map styles:

[Supported region](#)

[Localization support](#)

[Supported map styles](#)

Azure Maps geocoding coverage

11/2/2020 • 7 minutes to read • [Edit Online](#)

The Azure Maps [Search service](#) supports geocoding, which means that your API request can have search terms, like an address or the name of a place, and returns the result as latitude and longitude coordinates. For example, the Azure Maps [Get Search Address API](#) receives queries that contain location information, and returns results as latitude and longitude coordinates.

However, the Azure Maps [Search service](#) doesn't have the same level of information and accuracy for all regions and countries. Use this article to determine what kind of locations you can reliably search for in each region.

The ability to geocode in a country/region is dependent upon the road data coverage and geocoding precision of the geocoding service. The following categorizations are used to specify the level of geocoding support in each country/region.

- **Address points** - Address data can be resolved to latitude/longitude coordinates within the address parcel (property boundary). Address points are often referred to as being 'Rooftop' accurate, which is the highest level of accuracy available for addresses.
- **House numbers** - Addresses are interpolated to a latitude/longitude coordinate on the street.
- **Street level** - Addresses are resolved to the latitude/longitude coordinate of the street that contains the address. The house number may not be processed.
- **City level** - City place names are supported.

Americas

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Anguilla				✓	✓
Antarctica				✓	✓
Antigua and Barbuda			✓	✓	✓
Argentina	✓	✓	✓	✓	✓
Aruba				✓	✓
Bahamas			✓	✓	✓
Barbados			✓	✓	✓
Belize				✓	✓
Bermuda			✓	✓	✓
Bolivia			✓	✓	✓

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Bonaire, Sint Eustatius, and Saba				✓	✓
Brazil	✓	✓	✓	✓	✓
Canada	✓	✓	✓	✓	✓
Cayman Islands			✓	✓	✓
Chile	✓	✓	✓	✓	✓
Colombia	✓	✓	✓	✓	✓
Costa Rica			✓	✓	✓
Cuba			✓	✓	✓
Dominica			✓	✓	✓
Dominicana			✓	✓	✓
Ecuador			✓	✓	✓
El Salvador			✓	✓	✓
Falkland Islands				✓	✓
French Guiana			✓	✓	✓
Grenada			✓	✓	✓
Guadeloupe		✓	✓	✓	✓
Guam	✓	✓	✓	✓	✓
Guatemala			✓	✓	✓
Guyana				✓	
Haiti			✓	✓	✓
Honduras			✓	✓	✓
Jamaica			✓	✓	✓
Martinique		✓	✓	✓	✓
Mexico	✓	✓	✓	✓	✓

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Montserrat				✓	✓
Nicaragua			✓	✓	✓
Panama			✓	✓	✓
Paraguay		✓	✓	✓	✓
Peru		✓	✓	✓	✓
Puerto Rico	✓	✓	✓	✓	✓
Saint Barthélemy			✓	✓	✓
Saint Kitts and Nevis			✓	✓	✓
Saint Lucia				✓	✓
Saint Martin			✓	✓	✓
Saint Pierre and Miquelon			✓	✓	✓
Saint Vincent and the Grenadines				✓	✓
Sint Maarten			✓	✓	✓
South Georgia and the South Sandwich Islands				✓	✓
Suriname				✓	✓
Trinidad and Tobago			✓	✓	✓
United States Minor Outlying Islands				✓	✓
United States of America	✓	✓	✓	✓	✓
Uruguay	✓	✓	✓	✓	✓
Venezuela			✓	✓	✓

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
British Virgin Islands				✓	✓
U.S. Virgin Islands	✓	✓	✓	✓	✓

Asia Pacific

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
American Samoa			✓	✓	✓
Australia	✓	✓	✓	✓	✓
Bangladesh				✓	✓
Bhutan				✓	✓
British Indian Ocean Territory				✓	✓
Brunei	✓		✓	✓	✓
Cambodia				✓	✓
China				✓	✓
Christmas Island	✓		✓	✓	✓
Cocos (Keeling) Islands				✓	✓
Comoros				✓	✓
Cook Islands				✓	✓
Fiji				✓	✓
French Polynesia				✓	✓
Heard Island and McDonald Islands				✓	✓
Hong Kong SAR	✓	✓	✓	✓	✓
Indonesia	✓	✓	✓	✓	✓
India	✓	✓	✓	✓	

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Japan				✓	✓
Kiribati				✓	✓
Korea				✓	✓
Laos				✓	✓
Macao SAR	✓	✓	✓	✓	✓
Malaysia	✓	✓	✓	✓	✓
Micronesia				✓	✓
Mongolia				✓	✓
Nauru				✓	✓
Nepal				✓	✓
New Caledonia				✓	✓
New Zealand	✓	✓	✓	✓	✓
Niue				✓	✓
Norfolk Island				✓	✓
North Korea				✓	✓
Northern Mariana Islands			✓	✓	✓
Pakistan				✓	✓
Palau				✓	✓
Papua New Guinea				✓	✓
Philippines	✓	✓	✓	✓	✓
Pitcairn				✓	✓
Samoa				✓	✓
Senkaku Islands	✓			✓	✓
Singapore	✓	✓	✓	✓	✓

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Solomon Islands				✓	✓
Southern Kurils	✓			✓	✓
Sri Lanka				✓	✓
Taiwan	✓	✓	✓	✓	✓
Thailand	✓		✓	✓	✓
Tokelau				✓	✓
Tonga				✓	✓
Turks and Caicos Islands				✓	✓
Tuvalu				✓	✓
Vanuatu				✓	✓
Vietnam	✓	✓	✓	✓	✓
Wallis and Futuna				✓	✓

Europe

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Albania			✓	✓	✓
Andorra	✓	✓	✓	✓	✓
Armenia	✓	✓		✓	✓
Austria	✓	✓	✓	✓	✓
Azerbaijan	✓	✓		✓	✓
Belgium	✓	✓	✓	✓	✓
Bosnia And Herzegovina	✓	✓	✓	✓	✓
Bulgaria	✓	✓	✓	✓	✓
Belarus		✓	✓	✓	✓

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Croatia	✓	✓	✓	✓	✓
Cyprus		✓	✓	✓	✓
Czech Republic	✓	✓	✓	✓	✓
Denmark	✓	✓	✓	✓	✓
Estonia	✓	✓	✓	✓	✓
Faroe Islands				✓	✓
Finland	✓	✓	✓	✓	✓
France	✓	✓	✓	✓	✓
Georgia	✓	✓		✓	✓
Germany	✓	✓	✓	✓	✓
Gibraltar		✓	✓	✓	✓
Greece	✓	✓	✓	✓	✓
Greenland				✓	✓
Guernsey		✓	✓	✓	✓
Hungary	✓	✓	✓	✓	✓
Iceland	✓	✓	✓	✓	✓
Ireland	✓	✓	✓	✓	✓
Isle Of Man		✓	✓	✓	✓
Italy	✓	✓	✓	✓	✓
Jan Mayen	✓			✓	✓
Jersey		✓	✓	✓	✓
Kazakhstan	✓	✓	✓	✓	✓
Kosovo			✓	✓	✓
Kyrgyzstan				✓	✓
Latvia		✓	✓	✓	✓

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Liechtenstein	✓	✓	✓	✓	✓
Lithuania	✓	✓	✓	✓	✓
Luxembourg	✓	✓	✓	✓	✓
North Macedonia	✓	✓	✓	✓	✓
Malta		✓	✓	✓	✓
Moldova	✓	✓	✓	✓	✓
Monaco		✓	✓	✓	✓
Montenegro		✓	✓	✓	✓
Netherlands	✓	✓	✓	✓	✓
Norway	✓	✓	✓	✓	✓
Poland	✓	✓	✓	✓	✓
Portugal	✓	✓	✓	✓	✓
+Azores and Madeira			✓	✓	✓
Romania		✓	✓	✓	✓
Russian Federation	✓	✓	✓	✓	✓
San Marino	✓	✓	✓	✓	✓
Serbia	✓	✓	✓	✓	✓
Slovakia	✓	✓	✓	✓	✓
Slovenia	✓	✓	✓	✓	✓
Spain	✓	✓	✓	✓	✓
Svalbard	✓		✓	✓	✓
Sweden		✓	✓	✓	✓
Switzerland	✓	✓	✓	✓	✓
Tajikistan				✓	✓

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Turkey	✓	✓	✓	✓	✓
Turkmenistan				✓	✓
Ukraine	✓	✓	✓	✓	✓
United Kingdom	✓	✓	✓	✓	✓
Uzbekistan				✓	✓
Vatican City			✓	✓	✓

Middle East and Africa

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Afghanistan				✓	✓
Algeria			✓	✓	✓
Angola			✓	✓	✓
Bahrain	✓	✓	✓	✓	✓
Benin			✓	✓	✓
Botswana			✓	✓	✓
Bouvet Island				✓	✓
Burkina Faso			✓	✓	✓
Burundi			✓	✓	✓
Cameroon			✓	✓	✓
Cabo Verde			✓	✓	✓
Central African Republic			✓	✓	✓
Chad			✓	✓	✓
Congo			✓	✓	✓
Côte d'Ivoire			✓	✓	✓

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Democratic Republic of the Congo			✓	✓	✓
Djibouti			✓	✓	✓
Egypt	✓	✓	✓	✓	✓
Equatorial Guinea, Republic of			✓	✓	✓
Eritrea			✓	✓	✓
Ethiopia			✓	✓	✓
French Southern Territories				✓	✓
Gabon			✓	✓	✓
Gambia				✓	✓
Ghana			✓	✓	✓
Guinea			✓	✓	✓
Guinea-Bissau			✓	✓	✓
Iran				✓	✓
Iraq			✓	✓	✓
Israel	✓	✓		✓	✓
Jordan	✓	✓	✓	✓	✓
Kenya			✓	✓	✓
Kuwait	✓	✓	✓	✓	✓
Lebanon	✓		✓	✓	✓
Lesotho			✓	✓	✓
Liberia			✓	✓	✓
Libya			✓	✓	✓
Madagascar			✓	✓	✓

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Malawi			✓	✓	✓
Maldives				✓	✓
Mali			✓	✓	✓
Marshall Islands				✓	✓
Mauritania			✓	✓	✓
Mauritius			✓	✓	✓
Mayotte			✓	✓	✓
Morocco	✓	✓	✓	✓	✓
Mozambique			✓	✓	✓
Myanmar				✓	✓
Namibia		✓	✓	✓	✓
Niger			✓	✓	✓
Nigeria		✓	✓	✓	✓
Oman			✓	✓	✓
Qatar	✓		✓	✓	✓
Réunion		✓	✓	✓	✓
Rwanda			✓	✓	✓
Saint Helena				✓	✓
Saudi Arabia		✓	✓	✓	✓
Senegal			✓	✓	✓
Seychelles			✓	✓	✓
Sierra Leone			✓	✓	✓
Somalia				✓	✓
South Africa	✓	✓	✓	✓	✓
South Sudan			✓	✓	✓

COUNTRY/REGION	ADDRESS POINTS	HOUSE NUMBERS	STREET LEVEL	CITY LEVEL	POINTS OF INTEREST
Sudan			✓	✓	✓
Swaziland			✓	✓	✓
Syria				✓	✓
São Tomé and Príncipe			✓	✓	✓
Tanzania			✓	✓	✓
Togo			✓	✓	✓
Tunisia	✓		✓	✓	✓
Uganda			✓	✓	✓
United Arab Emirates	✓	✓	✓	✓	✓
Yemen				✓	✓
Zambia			✓	✓	✓
Zimbabwe			✓	✓	✓

Next steps

Learn more about Azure Maps geocoding:

[Azure Maps Search service](#)

Azure Maps traffic coverage

11/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Maps provides rich traffic information in the form of **traffic flow** and **incidents**. This data can be visualized on maps or used to generate smarter routes that factor in real driving conditions.

However, Maps doesn't have the same level of information and accuracy for all countries or regions. The following table provides information about what kind of traffic information you can request from each country or region:

Americas

COUNTRY/REGION	INCIDENTS	FLOW
Argentina	✓	✓
Brazil	✓	✓
Canada	✓	✓
Chile	✓	✓
Colombia	✓	✓
Mexico	✓	✓
Peru	✓	✓
United States	✓	✓
+Puerto Rico	✓	✓
Uruguay	✓	✓

Asia Pacific

COUNTRY/REGION	INCIDENTS	FLOW
Australia	✓	✓
Brunei	✓	✓
Hong Kong SAR	✓	✓
India	✓	✓
Indonesia	✓	✓

COUNTRY/REGION	INCIDENTS	FLOW
Kazakhstan	✓	✓
Macao SAR	✓	✓
Malaysia	✓	✓
New Zealand	✓	✓
Philippines	✓	✓
Singapore	✓	✓
Taiwan	✓	✓
Thailand	✓	✓
Vietnam	✓	✓

Europe

COUNTRY/REGION	INCIDENTS	FLOW
Andorra	✓	✓
Austria	✓	✓
Belarus	✓	✓
Belgium	✓	✓
Bosnia and Herzegovina	✓	✓
Bulgaria	✓	✓
Croatia	✓	✓
Czech Republic	✓	✓
Denmark	✓	✓
Estonia		✓
Finland	✓	✓
+Åland Islands	✓	✓
France	✓	✓
Monaco	✓	✓

COUNTRY/REGION	INCIDENTS	FLOW
Germany	✓	✓
Greece	✓	✓
Hungary	✓	✓
Iceland	✓	✓
Ireland	✓	✓
Italy	✓	✓
Kazakhstan	✓	✓
Latvia	✓	✓
Lesotho	✓	✓
Liechtenstein	✓	✓
Lithuania	✓	✓
Luxembourg	✓	✓
Malta	✓	✓
Monaco	✓	✓
Netherlands	✓	✓
Norway	✓	✓
Poland	✓	✓
Portugal	✓	✓
+Azores and Madeira	✓	✓
Romania	✓	✓
Russian Federation	✓	✓
San Marino	✓	✓
Serbia	✓	✓
Slovakia	✓	✓
Slovenia	✓	✓

COUNTRY/REGION	INCIDENTS	FLOW
Spain	✓	✓
+Andorra	✓	✓
+Balearic Islands	✓	✓
+Canary Islands	✓	✓
Sweden	✓	✓
Switzerland	✓	✓
Turkey	✓	✓
Ukraine	✓	✓
United Kingdom	✓	✓
+Gibraltar	✓	✓
+Guernsey & Jersey	✓	✓
+Isle of Man	✓	✓
Vatican City	✓	✓

Middle East and Africa

COUNTRY/REGION	INCIDENTS	FLOW
Bahrain	✓	✓
Egypt	✓	✓
Israel	✓	✓
Kenya	✓	✓
Kuwait	✓	✓
Morocco	✓	✓
Mozambique	✓	✓
Nigeria	✓	✓
Oman	✓	✓
Qatar	✓	✓

COUNTRY/REGION	INCIDENTS	FLOW
Saudi Arabia	✓	✓
South Africa	✓	✓
United Arab Emirates	✓	✓

Next steps

For more information about Azure Maps traffic data, see the [Traffic](#) reference pages.

Azure Maps render coverage

11/2/2020 • 4 minutes to read • [Edit Online](#)

Azure Maps uses both raster tiles and vector tiles to create maps. At the lowest resolution, the entire world fits in a single tile. At the highest resolution, a single tile represents 38 square meters. You'll see more details about continents, regions, cities, and individual streets as you zoom in the map. For more information about tiles, see [Zoom levels and tile grid](#).

However, Maps doesn't have the same level of information and accuracy for all regions. The following tables detail the level of information you can render for each region.

Legend

SYMBOL	MEANING
✓	Region is represented with detailed data.
Ø	Region is represented with simplified data.

Africa

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Algeria	✓	✓
Angola	✓	✓
Benin	✓	✓
Botswana	✓	✓
Burkina Faso	✓	✓
Burundi	✓	✓
Cabo Verde	✓	✓
Cameroon	✓	✓
Central African Republic	✓	Ø
Chad	✓	Ø
Comoros	✓	Ø
Democratic Republic of the Congo	✓	✓
Côte d'Ivoire	✓	Ø

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Djibouti	✓	∅
Egypt	✓	✓
Equatorial Guinea	✓	∅
Eritrea	✓	∅
Ethiopia	✓	∅
Gabon	✓	✓
Gambia	✓	∅
Ghana	✓	✓
Guinea	✓	∅
Guinea-Bissau	✓	∅
Kenya	✓	✓
Lesotho	✓	✓
Liberia	✓	∅
Libya	✓	∅
Madagascar	✓	∅
Malawi	✓	✓
Mali	✓	✓
Mauritania	✓	✓
Mauritius	✓	✓
Mayotte	✓	✓
Morocco	✓	✓
Mozambique	✓	✓
Namibia	✓	✓
Niger	✓	✓
Nigeria	✓	✓

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Réunion	✓	✓
Rwanda	✓	✓
Saint Helena, Ascension and Tristan da Cunha	✓	∅
São Tomé and Príncipe	✓	∅
Senegal	✓	✓
Sierra Leone	✓	✓
Somalia	✓	✓
South Africa	✓	✓
South Sudan	✓	✓
Sudan	✓	✓
Swaziland	✓	✓
United Republic of Tanzania	✓	✓
Togo	✓	✓
Tunisia	✓	✓
Uganda	✓	✓
Zambia	✓	✓
Zimbabwe	✓	✓

Americas

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Anguilla	✓	✓
Antigua and Barbuda	✓	✓
Argentina	✓	✓
Aruba	✓	✓
Bahamas	✓	✓
Barbados	✓	✓

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Belize	✓	✓
Bermuda	✓	✓
Plurinational State of Bolivia	✓	✓
Bonaire, Sint Eustatius, and Saba	✓	✓
Brazil	✓	✓
Canada	✓	✓
Cayman Islands	✓	✓
Chile	✓	✓
Colombia	✓	✓
Costa Rica	✓	✓
Cuba	✓	✓
Curaçao	✓	✓
Dominica	✓	✓
Dominican Republic	✓	✓
Ecuador	✓	✓
Falkland Islands (Malvinas)	✓	✓
French Guiana	✓	✓
Greenland	✓	∅
Grenada	✓	✓
Guadeloupe	✓	✓
Guatemala	✓	✓
Guyana	✓	✓
Haiti	✓	✓
Honduras	✓	✓
Jamaica	✓	✓

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Martinique	✓	✓
Mexico	✓	✓
Montserrat	✓	✓
Nicaragua	✓	✓
Northern Mariana Islands	✓	✓
Panama	✓	✓
Paraguay	✓	✓
Peru	✓	✓
Puerto Rico	✓	✓
Quebec (Canada)	✓	✓
Saint Barthélemy	✓	✓
Saint Kitts and Nevis	✓	✓
Saint Lucia	✓	✓
Saint Martin (French)	✓	✓
Saint Pierre and Miquelon	✓	✓
Saint Vincent and the Grenadines	✓	✓
Sint Maarten (Dutch)	✓	✓
South Georgia and the South Sandwich Islands	✓	✓
Suriname	✓	✓
Trinidad and Tobago	✓	✓
Turks and Caicos Islands	✓	✓
United States	✓	✓
Uruguay	✓	✓
Venezuela	✓	✓
Virgin Islands, British	✓	✓

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Virgin Islands, U.S.	✓	✓

Asia

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Afghanistan		∅
Bahrain	✓	✓
Bangladesh		∅
Bhutan		∅
British Indian Ocean Territory		∅
Brunei	✓	✓
Cambodia		∅
China		∅
Cocos (Keeling) Islands		∅
Democratic People's Republic of Korea		∅
Hong Kong SAR	✓	✓
India	∅	✓
Indonesia	✓	✓
Iran		∅
Iraq	✓	✓
Israel		✓
Japan		∅
Jordan	✓	✓
Kazakhstan		✓
Kuwait	✓	✓
Kyrgyzstan		∅
Lao People's Democratic Republic		∅

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Lebanon	✓	✓
Macao SAR	✓	✓
Malaysia	✓	✓
Maldives		∅
Mongolia		∅
Myanmar		∅
Nepal		∅
Oman	✓	✓
Pakistan		∅
Philippines	✓	✓
Qatar	✓	✓
Republic of Korea	✓	∅
Saudi Arabia	✓	✓
Senkaku Islands		✓
Singapore	✓	✓
Sri Lanka		∅
Syrian Arab Republic		∅
Taiwan	✓	✓
Tajikistan		∅
Thailand	✓	✓
Timor-Leste		∅
Turkmenistan		∅
United Arab Emirates	✓	✓
United States Minor Outlying Islands		∅
Uzbekistan		∅

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Vietnam	✓	✓
Yemen	✓	✓

Oceania

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
American Samoa		✓
Australia	✓	✓
Cook Islands		∅
Fiji		∅
French Polynesia		∅
Guam	✓	✓
Kiribati		∅
Marshall Islands		∅
Micronesia		∅
Nauru		∅
New Caledonia		∅
New Zealand	✓	✓
Niue		∅
Norfolk Island		∅
Palau		∅
Papua New Guinea		∅
Pitcairn		∅
Samoa		∅
Solomon Islands		∅
Tokelau		∅
Tonga		∅

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Tuvalu		∅
Vanuatu		∅
Wallis and Futuna		∅

Europe

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Albania	✓	✓
Andorra	✓	✓
Armenia	✓	∅
Austria	✓	✓
Azerbaijan	✓	∅
Belarus	∅	✓
Belgium	✓	✓
Bosnia-Herzegovina	✓	✓
Bulgaria	✓	✓
Croatia	✓	✓
Cyprus	✓	✓
Czech Republic	✓	✓
Denmark	✓	✓
Estonia	✓	✓
Faroe Islands	✓	∅
Finland	✓	✓
France	✓	✓
Georgia	✓	∅
Germany	✓	✓
Gibraltar	✓	✓

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Greece	✓	✓
Guernsey	✓	✓
Hungary	✓	✓
Iceland	✓	✓
Ireland	✓	✓
Isle of Man	✓	✓
Italy	✓	✓
Jan Mayen	✓	✓
Jersey	✓	✓
Latvia	✓	✓
Liechtenstein	✓	✓
Lithuania	✓	✓
Luxembourg	✓	✓
North Macedonia	✓	✓
Malta	✓	✓
Moldova	✓	✓
Monaco	✓	✓
Montenegro	✓	✓
Netherlands	✓	✓
Norway	✓	✓
Poland	✓	✓
Portugal	✓	✓
Romania	✓	✓
Russian Federation	✓	✓
San Marino	✓	✓

COUNTRY/REGION	RASTER TILES UNIFIED	VECTOR TILES UNIFIED
Serbia	✓	✓
Slovakia	✓	✓
Slovenia	✓	✓
Southern Kurils	✓	✓
Spain	✓	✓
Svalbard	✓	✓
Sweden	✓	✓
Switzerland	✓	✓
Turkey	✓	✓
Ukraine	✓	✓
United Kingdom	✓	✓
Vatican City	✓	✓

Next steps

For more information about Azure Maps rendering, see [Zoom levels and tile grid](#).

Learn about the [coverage areas for the Maps routing service](#).

Azure Maps routing coverage

3/31/2021 • 3 minutes to read • [Edit Online](#)

This article provides coverage information for Azure Maps routing. Upon a search query, Azure Maps returns an optimal route from location A to location B. You are provided with accurate travel times, live updates of travel information, and route instructions. You can also add additional search parameters such as current traffic, vehicle type, and conditions to avoid. The optimization of the route depends on the region. That's because, Azure Maps has various levels of information and accuracy for different regions. The following table lists the regions and what kind of information you can request for them.

Check out coverage for [Geocoding](#).

Check out coverage for [Traffic](#).

Check out coverage for [Render](#).

Legend

SYMBOL	MEANING
✓	Country/region provided with detailed data.
Ø	Country/region provided with simplified data.
Country is missing	Country/region data is not provided.

The following table provides coverage information for Azure Maps routing.

Africa

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Algeria	✓		
Angola	✓		
Benin	✓		
Botswana	✓		
Burkina Faso	✓		
Burundi	✓		
Cameroon	✓		
Cabo Verde	✓		
Congo- Brazzaville Kinshasa	✓		

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Congo- Kinshasa	✓		
côte d'ivoire	✓		
Egypt	✓	✓	
Gabon	✓		
Gambia	✓		
Ghana	✓		
Kenya	✓	✓	
Lesotho	Ø	✓	
Malawi	✓		
Mali	✓		
Mauritania	✓		
Mauritius, Mayotte, and Réunion	✓		
Morocco	✓		
Mozambique	✓	✓	
Namibia	✓		
Niger	✓		
Nigeria	✓	✓	
Rwanda	✓		
Senegal	✓		
Seychelles	✓		
South Africa	✓	✓	
Swaziland	✓		
Tanzania	✓		
Togo	✓		
Tunisia	✓		

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Uganda	✓		
Zambia	✓		
Zimbabwe	✓		

Americas

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Argentina	✓	✓	✓
Antigua and Barbuda	✓		
Bahamas	✓		
Barbados	✓		
Belize	✓		
Brazil	✓	✓	✓
Canada	✓	✓	✓
Chile	✓	✓	✓
Colombia	✓	✓	
Costa Rica	✓		
Cuba	✓		
Dominica	✓		
Dominican Republic	✓		
Ecuador	✓		
El Salvador	✓		
French Guiana	✓		
Grenada	✓		
Guatemala	✓		
Guyana	✓		
Haiti	✓		

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Honduras	✓		
Jamaica	✓		
Mexico	✓	✓	✓
Nicaragua	✓		
Panama	✓		
Paraguay	✓		
Peru	✓	✓	
St. Kitts and Nevis	✓		
St. Lucia	✓		
St. Vincent & Grenadines	✓		
Suriname	✓		
Trinidad & Tobago	✓		
United States	✓	✓	✓
+American Samoa	✓		
+Northern Mariana Islands	✓		
+Puerto Rico	✓		
+U.S. Virgin Islands	✓		
Uruguay	✓	✓	✓
Venezuela	✓		

Asia Pacific

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Australia	✓	✓	✓
+Christmas Island	✓		
Brunei	✓	✓	
Cambodia	✓		

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Fiji	✓		
Guam	✓		
Hong Kong SAR	✓	✓	
India	✓	✓	
Indonesia	✓	✓	
Kazakhstan	✓		
Korea	Ø		
Laos	✓		
Macao SAR	✓	✓	
Malaysia	✓	✓	
Myanmar	✓		
New Zealand	✓	✓	✓
Philippines	✓	✓	
Singapore	✓	✓	
Taiwan	✓	✓	
Thailand	✓	✓	
Vietnam	✓	✓	

Europe

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Albania	✓		
Andorra	✓	✓	
Austria	✓	✓	✓
Belarus	✓		
Belgium	✓	✓	✓
Bolivia	✓		

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Bosnia-Herzegovina	✓		
Bulgaria	✓	✓	✓
Croatia	✓	✓	✓
Cyprus	✓		
Czech Republic	✓	✓	✓
Denmark	✓	✓	✓
+Faroe Islands	✓		
Estonia	✓	✓	✓
Finland	✓	✓	✓
France	✓	✓	✓
+Guadeloupe	✓		
+Martinique	✓		
+St. Barthélemy	✓		
+St. Martin	✓		
+St. Pierre & Miquelon	✓		
Georgia	✓		
Germany	✓	✓	✓
Greece	✓	✓	✓
Guernsey	✓		
Hungary	✓	✓	✓
Iceland	✓	✓	
Ireland	✓	✓	✓
Italy	✓	✓	✓
Jersey	✓		
Latvia	✓	✓	✓

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Liechtenstein	✓	✓	
Lithuania	✓	✓	✓
Luxembourg	✓	✓	✓
North Macedonia	✓		
Malta	✓	✓	
Moldova	✓		
Monaco	✓	✓	
Montenegro	✓		✓
Netherlands	✓	✓	✓
+Aruba	✓		
+Caribbean Netherlands	✓		
+Curaçao	✓		
+Sint Maarten	✓		
Norway	✓	✓	✓
Poland	✓	✓	✓
Portugal	✓	✓	✓
Romania	✓	✓	✓
Russian Federation	✓	✓	✓
San Marino	✓	✓	
Serbia	✓		✓
Slovakia	✓	✓	✓
Slovenia	✓	✓	✓
Spain	✓	✓	✓
Sweden	✓	✓	✓
Switzerland	✓	✓	✓

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Turkey	✓	✓	✓
Ukraine	✓	✓	
United Kingdom	✓	✓	✓
+Anguilla	✓		
+Bermuda	✓		
+British Virgin Islands	✓		
+Cayman Islands	✓		
+Gibraltar	✓	✓	
+Montserrat	✓		
+Turks and Caicos Islands	✓		
Vatican City	✓	✓	
Isle of Man	✓		

Middle East

COUNTRY/REGION	ROUTING	ROUTING WITH TRAFFIC	TRUCK ROUTING
Bahrain	✓	✓	
Iraq	✓		
Israel	✓	✓	
Jordan	✓		
Kuwait	✓	✓	
Lebanon	✓		
Oman	✓	✓	
Qatar	✓	✓	
Saudi Arabia	✓	✓	
United Arab Emirates	✓	✓	
Yemen	✓		

Next steps

For more information about Azure Maps routing, see the [Routing](#) reference pages.

Azure Maps Mobility services (Preview) coverage

3/5/2021 • 10 minutes to read • [Edit Online](#)

IMPORTANT

Azure Maps Mobility services are currently in public preview. This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

The Azure Maps [Mobility services](#) improves the development time for applications with public transit features, such as transit routing and search for nearby public transit stops. Users can retrieve detailed information about transit stops, lines, and schedules. The Mobility services also allow users to retrieve stop and line geometries, alerts for stops, lines, and service areas, and real-time public transit arrivals and service alerts. Additionally, the Mobility services provide routing capabilities with multimodal trip planning options. Multimodal trip planning incorporates walking, bicycling, and public transit options, all into one trip. Users can also access detailed multimodal step-by-step itineraries.

Azure Maps doesn't provide the same level of information and accuracy for all cities and countries/regions. The ability to call public transit data depends on the metro area. In addition, the map data may not include all public transit options and agencies that serve the metro area.

The following table provides coverage information for Azure Maps Mobility services.

SYMBOL	MEANING
*	Nearly full coverage for the country/region.

Americas

COUNTRY/REGION	CITY (METRO AREA)
Antigua and Barbuda	Antigua and Barbuda*
Argentina	Azul, Bahía Blanca, Buenos Aires, Caleta Olivia, Catamarca, Chivilcoy, Comodoro Rivadavia, Concordia, Córdoba, Corrientes, General Pico, Gualeguaychu, La Rioja, Mar del Plata, Mendoza, Miramar, Necochea, Neuquén, Oberá, Olavarría, Paraná, Posadas, Rafaela, Rio Tercero, Rosario, Salta, San Carlos de Bariloche, San Luis, San Miguel de Tucumán, San Pedro, Santa Fe, Tandil, Ushuaia, Victoria, Viedma, Villa María
Barbados	Barbados*

COUNTRY/REGION	CITY (METRO AREA)
Brazil	Angra dos Reis, Anápolis, Apucarana, Aracaju, Araraquara, Araxá, Araçatuba, Atibaia, Bage, Barretos, Bauru, Bebedouro, Belém, Belo Horizonte, Blumenau, Boa Vista, Botucatu, Brasília, Caldas Novas, Campina Grande, Campinas, Campo Belo, Campo Grande, Caraguatatuba, Caratinga, Cascavel, Cataguases, Caxias, Leopoldina e Região, Catalão, Caxias do Sul, Chapecó, Cianorte, Conselheiro Lafaiete, Corumbá, Criciúma, Cruzeiro do Sul, Cuiabá, Curitiba, Curitibanos, Curvelo, Diamantina, Divinópolis, Dourados, Estrela, Feira de Santana, Fernando de Noronha, Florianópolis, Fortaleza, Foz do Iguaçu, Franca, Garanhuns, Goiania, Governador Valadares, Guarapuava, Imperatriz, Ipatinga, Iratí, Itabira, Itabuna, Itajaí, Itajubá, Ituiutaba, Jaguaraو, Jaraguá do Sul, João Pessoa, Joinville, Juazeiro do Norte, Juiz de Fora, Jundiaí, Lages, Lavras e Regiao, Lucas do Rio Verde, Londrina, Macapá, Macaé, Maceió, Mafra e Rio Negro, Manaus, Manhuacu, Maringá, Marília, Monte Carmelo, Montes Claros, Mossoró, Natal, Osorio, Ourinhos, Ouro Preto, Palmas, Paracatu, Paranaguá, Parnaíba, Passo Fundo, Passos, Pato Branco, Patos de Minas, Patrocínio, Pelotas, Picos, Piracicaba, Pirapora, Pocos de Caldas, Ponta Grossa, Porto Alegre, Porto Ferreira, Porto Seguro, Porto Velho, Praia Grande, Recife, Ribeirão Preto, Rio, Rio Branco, Rio Verde, Rondonópolis, Salinas, Salvador, Santa Cruz do Sul, Santa Maria, Santa Rita do Sapucaí, Santarem, Santiago del Estero, Santos, São Gabriel do Oeste, São João del Rei, Tiradentes e Regiao, São Jose do Rio Preto, São Mateus, São Paulo, Sorocaba, São Carlos, São Francisco do Sul, São José dos Campos, São Lourenço, São Luís, Taubaté, Telesmaco Borba, Teófilo Otoni, Teresina, Toledo , Três Lagoas, Tucurui, Ubatuba, Uberaba, Uberlândia, Ubá, Uruguaiana, Varginha, Vícose, Videira & Fraiburgo, Vitória, Vitória da Conquista, Volta Redonda, Votuporanga
Canada	Banff (AB), Brandon (MB), Calgary (AB), Chatham-Kent (ON), Comox Valley (BC), Cowichan Valley (BC), Cranbrook (BC), Edmonton (AB), Fort St. John, Fredericton (NB), Greater Sudbury (ON), Greater Vancouver (BC), Halifax (NS), Kamloops (BC), Kelowna - Vernon (BC), Kingston (ON), London (ON), Moncton (NB), Montreal (QC), Nanaimo (BC), Ottawa (ON), Prince George (BC), Québec City (QC), Red Deer (AB), Regina (SK), Rimouski (QC), Saskatoon (SK), Sherbrooke (QC), Southwest British Columbia (BC), Squamish (BC), St. John's (NL), Sunshine Coast, Thunder Bay (ON), Toronto (ON), Victoria (BC), West Kootenay (BC), Whistler (BC), Windsor (ON), Winnipeg (MB), Woodstock
Chile	Antofagasta, Arica, Aysén, Chillán, Concepción, Constitución, Copiapó, Curicó, Iquique, La Serena y Coquimbo, Linares, Los Angeles (Chile), Los Lagos, Punta Arenas, Rancagua, Santiago, Talca, Temuco, Valdivia, Valparaíso, Viña del Mar

COUNTRY/REGION	CITY (METRO AREA)
Colombia	Barranquilla, Bogotá, Bucaramanga, Cali, Cartagena, Ibagué, Medellín, Pasto, Popayán, Santa Marta, Sincelejo, Valledupar
Costa Rica	San José
Dominican Republic	Santo Domingo
Ecuador	Cuenca, Guayaquil, Loja, Manta, Milagro
El Salvador	San Salvador
Guatemala	Ciudad de Guatemala (GT)
Mexico	Acapulco, Aguascalientes, Cancun, Durango, Mexico City, Guadalajara, Lion, Merida, Monterrey, Puebla, Puerto Vallarta, Querétaro, San Luis Potosi, Tijuana, Torreon
Nicaragua	Managua
Panama	Panama*
Peru	Cusco, Lima
Puerto Rico	San Juan
Suriname	Paramaribo
Uruguay	Montevideo, Paysandu, Punta del Este, Salto

COUNTRY/REGION	CITY (METRO AREA)
United States of America	Albany (GA), Albany (NY), Albuquerque (NM), Anchorage (AK), Ann Arbor (MI) Appleton-Oshkosh-Neenah (WI), Asheville (NC), Athens (GA), Athens (OH), Atlanta (GA), Austin (TX), Bakersfield (CA), Baltimore (MD), Bend-Redmond (OR), Berkshire County (MA), Birmingham (AL), Bloomington (IN), Boise (ID), Boston (MA), Boulder (CO), Bowling Green (KY), Brevard County (FL), Buffalo (NY), Butte (MT), Cape Cod (MA), Centre County (PA), Champaign-Urbana (IL), Charleston (SC), Charleston (WV), Charlotte (NC), Charlottesville (VA), Chattanooga (TN), Cheyenne (WY), Chicago (IL), Cincinnati (OH), Citrus County (FL), Cleveland (OH), Coachella Valley (CA), Colorado Springs (CO), Columbia (TN), Columbia (SC), Columbus (OH), Corpus Christi (TX), Dallas/Forth Worth (TX), Dayton (OH), Delaware, Denver (CO), Des Moines (IA), Detroit (MI), Duluth (MN), El Paso (TX), Eugene (OR), Fairbanks (AK), Fargo (ND), Fayetteville (NC), Flagstaff (AZ), Flint (MI) Fort Collins (CO), Fort Wayne (IN), Fresno (CA), Gainesville (FL), Grand Forks (ND), Grand Rapids (MI), Green Bay (WI), Greensboro (NC), Greenville (SC), Gunnison (CO), Hampton Roads (VA), Hanford (CA), Hartford (CT), Hernando County (FL), Hinesville (GA), Honolulu (HI), Houston (TX), Humboldt County (CA), Huntsville (AL), Indianapolis (IN), Ithaca (NY), Jackson (MS), Jackson (TN), Jacksonville - St. John's County (FL), Johnson city (TN), Jonesboro (AR), Joplin (MO), Juneau (AK), Kalamazoo (MI), Kalispell (MT), Kansas City (MO), Kauai (HI), Ketchum (ID), Knoxville (TN), Lafayette (IN), Lancaster (PA), Lansing (MI), Laredo (TX), Las Vegas (NV), Lawrence (KS), Lee County (FL), Lexington (KY), Lincoln County (OR), Little Rock (AR), Los Angeles (CA), Louisville (KY), Lubbock (TX), Madison (WI), Manchester (NH), McAllen (TX), Memphis (TN), Miami (FL), Milwaukee-Waukesha (WI), Minneapolis-St. Paul (MN), Missoula (MT), Modesto (USA), Moline (IL), Monroe County (PA), Montgomery (AL), Morgantown (WV), Nashville (TN), Navajo Nation, New Haven (CT), New Orleans (LA), NYC-NJ Area(NY), Ocala (FL), Okaloosa County (FL), Oklahoma City (OK), Omaha (NE), Orlando (FL), Palm Desert (CA), Panama City (FL), Pensacola (FL), Peoria (IL), Philadelphia (PA), Phoenix (AZ), Pittsburgh (PA), Portland (ME), Portland (OR), Racine (WI), Raleigh (NC), Redding (CA), Reno & Lake Tahoe (NV), Richmond (VA), Roanoke Valley (VA - Lynchburg), Rochester (NY), Rockford (IL), Rocky Mount (NC), Rocky Mountain National Park (CO), Rogue Valley (OR), Roseburg (OR), Roseville (CA), Sacramento (CA), Salem (OR), Salt Lake City (UT), San Antonio (TX), San Diego (CA), San Luis Obispo (CA), Santa Barbara (CA), Santa Fe (NM), Sarasota (FL), Savannah (GA), Seacoast Region (NH), Seattle-Tacoma-Bellevue (WA), SF Bay Area (CA), SF-San Jose Area (CA), Sioux City (IA), Sioux Falls (SD), Sitka (AK), Spokane (WA), Springfield (MA), South Bend (IN), Springfield (IL), Springfield (Mass), St. George (UT), St. Louis (MO), Stockton (CA), Syracuse-Utica (NY), Tallahassee (FL), Tampa-St. Petersburg (FL), Terre Haute (IN), Toledo (OH), Topeka (KS), Traverse City (MI), Tucson (AZ), Tulsa (OK), Vermont, Victorville (CA), Volusia County (FL), Waco (TX), Washington (DC), Waterbury (CT), Wichita (KS), Wichita Falls (TX) Wilmington (NC), Yakima (WA), Youngstown (OH), York County (PA), Yuma County (AZ)

COUNTRY/REGION +U.S. Virgin Islands	CITY (METRO AREA) U.S. Virgin Islands*
Venezuela	Caracas

Asia Pacific

COUNTRY/REGION	CITY (METRO AREA)
Australia	Adelaide, Alice Springs, Bowen, Brisbane, Bundaberg, QLD, Burnie, Cairns, Canberra, Darwin, Gladstone, Hobart, Innisfail, Launceston, Mackay, Magnetic Island, Maryborough-Hervey Bay, Melbourne, New South Wales, Perth, Rockhampton, South East Queensland, Sydney, Toowoomba, Townsville, Victoria, Warwick, Yeppoon
Brunei	Bandar Seri Begawan
China	Changchun, Changsha, Chengdu, Chongqing, Dalian, Datong, Dongguan, Hangzhou, Harbin, Jiangyin, Jinan, Nanjing, Nantong, Ningbo, Pingdingshan, Qingdao, Shenyang, Suzhou, Tangshan, Tianjin, Weifang, Wuhan, Wuxi, Yantai, Yixing, Zhuhai, Shanghai, Beijing, Guangzhou, Shenzhen, Zhengzhou
Hong Kong SAR	Hong Kong SAR*
Macao SAR	Macao SAR*
Maldives	Male
India	Ahmedabad, Bengaluru, Delhi, Hyderabad, Mumbai, Mysuru, Pune
Indonesia	Bandung, Banjarmasin, Banyuwangi, Batam, Denpasar, Jakarta, Kediri, Malang, Palembang, Semarang, Surabaya, Surakarta, Yogyakarta
Japan	Hokkaido, Shizuoka Prefecture, Tokyo, Wakkanai, Yamanashi Prefecture
Malaysia	Ipooh, Johor Bahru, Kuala Lumpur, Kuantan, Penang
New Zealand	Auckland, Christchurch, Dunedin, Queenstown, Timaru, Wellington
Philippines	Manila
Singapore	Singapore*
South Korea	Busan, Seoul

COUNTRY/REGION	CITY (METRO AREA)
Taiwan	Changhua County, Taipei
Thailand	Bangkok, Chiang Mai
Uzbekistan	Samarkand
Vietnam	Hanoi, Ho Chi Minh City

Europe

COUNTRY/REGION	CITY (METRO AREA)
Andorra	Andorra la Vella
Austria	Vienna
Belarus	Gomel, Grodno, Polotsk & Novopolotsk, Zhlobin, Vileyka, Maladziečna, Minsk, Rechytsha
Belgium	Belgium*
Bolivia	La Paz, Santa Cruz de la Sierra
Bosnia and Herzegovina	Sarajevo
Bulgaria	Balchik, Blagoevgrad, Burgas, Dobrich, Gabrovo, Haskovo, Kardzhali, Lovech, Nessebar, Pazardzhik, Pernik, Pleven, Plovdiv, Ruse, Shumen, Sliven, Stara Zagora, Vratsa, Yambol, Varna, Veliko, Sofia
Croatia	Crikvenica, Dubrovnik, Rijeka, Slovanski Brod, Zagreb
Cyprus	Larnaca, Limassol, Nicosia
Czech Republic	Brno, Jablonec, Karlovy Vary, Liberec, Ostrava, Prague
Denmark	Denmark*
Estonia	Estonia*
Finland	Hämeenlinna, Helsinki, Joensuu, Jyväskylä, Kajaani, Kouvola - Kotka, Kuopio, Lappeenranta, Mikkeli, Oulu, Pori, Rovaniemi, Seinäjoki, Tampere, Turku, Vaasa

COUNTRY/REGION	CITY (METRO AREA)
France	Amberieu-en-Bugey, Amiens, Angers, Annecy, Annonay, Arras, Aubenas, Bayonne, Besançon, Blois, Bordeaux, Boulogne sur Mer, Brest, Briançon, cannes, Châlons-en-Champagne, Chartres, Clermont-Ferrand, Colmar, Côte d'Azur, Dax, Dijon, Grenoble, Haguenau, La Rochelle, Le Mans, Lens, Lille, Lorient, Lyon, MACS, Marseille & Provence, Metz, Millau, Mont-de-Marsan, Montpellier, Mulhouse, Nancy, Nantes, Nice, Nice Côte d'Azur, Nîmes, Normandy, Nyons, Paris, Poitiers, Privas, Quimper, Rennes, Saint Malo, Saint-Étienne, Saint-Nazaire, Saintes, Sarrebourg, Sete, Strasbourg, Tarbes, Toulouse, Tours
+ French Guiana	Cayenne
+ New Caledonia	Nouméa
Georgia	Tbilisi
Germany	Berlin, Brandenburg, Bremen & Niedersachsen, Cologne, Eisenach, Frankfurt, Hamburg, Karlsruhe, Mainz, München - Munich, Rhein-Neckar Region, Rhein-Ruhr Region, Stuttgart, Titisee-Neustadt, Ulm
Greece	Agrinio, Aigio, Athens, Arta, Amorgos, Chania, Corfu, Chios Kos, Heraklion, Ioannina, Kavala, Kalamata, Khios, Komotini, Kos, Larissa, Meganisi, Milos, Mykonos, Patra, Rethimno, Rhodes, Santorini, Serres, Syros, Tinos, Thessaloniki, Veria, Volos, Xanthi
Hungary	Budapest, Gyor, Miskolc, Nograd County, Pesc, Szeged, Székesfehérvár
Iceland	Ísland - Iceland*
Ireland	Ireland*
Italy	Agrigento, Alessandria, Ancona, Bari, Bologna - Bologne, Cagliari - Sardinia, Campobasso, Catania e Messina, Cosenza, Crema, Cremona, Crotone, Cuneo, Firenze - Florence, Foggia, Genova - Genoa, Iglesias, La Spezia, Lecce, Matera, Milano - Milan, Napoli - Naples, Padova, Palermo, Parma, Perugia, Pescara, Pisa, Potenza, Roma - Rome, Siena e Grosseto, Siracusa - Syracuse, Taranto, Torino - Turin, Trento, Trieste, Udine, Venezia - Venice,
Latvia	Riga
Liechtenstein	Liechtenstein*
Lithuania	Druskininkai, Kauno, Klaipėda, Panevėžys, Vilnius
Luxembourg	Luxembourg*

COUNTRY/REGION	CITY (METRO AREA)
Moldova	Chisinau
Montenegro	Podgorica
Netherlands	Netherlands*
Norway	Norway*
Poland	Wrocław, Białystok, Bydgoszcz, Elbląg, Elk, Gorzow, Kętrzyna, Krakow, Leszno, Lodz, Lublin, Mrągowo, Olsztyn, Poznań, Rzeszów, Sanok, Starachowice, Świdnica, Szczecin, Tricity, Warsaw, Wodzisław Śląski, Wrocław, Zakopane
Portugal	Bragança, Coimbra, Funchal, Leiria, Lisboa, Portimao, Porto
Malta	Malta*
Romania	Alba Iulia, Arad, Bistrița, Brăila, Brașov, Bucharest, Buzau, Cluj Napoca, Constanța, Craiova, Deva, Focșani, Galati, Iași, Miercurea Ciuc, Oradea, Piatra Neamt, Pitești, Ploiești, Reșița, Satu Mare, Sibiu, Suceava, Targu Mures, Timisoara, Tulcea, Zalau
Russia	Kaliningrad, Rostov-on-Don, Volgograd, Yekaterinburg, Kazan, Kirov, Krasnodar, Moscow, Nalchik, Nizhny Novgorod, Novosibirsk, Noyabrsk, Omsk, Oryol, Perm, St Petersburg, Pyatigorsk, Tver, Tomsk
Serbia	Beograd, Kragujevac, Nis, Novi Sad, Valjevo, Subotica
Slovakia	Banská Bystrica, Bratislava, Košice, Presov, Prievidza, Ruzomberok a Liptovsky Mikulas, Stará Ľubovňa, Trencín
Slovenia	Koper, Ljubljana
Spain	Albacete, Corunna, Alicante, Almería, Asturias, Avila, Badajoz, Bay of Cadiz, Barcelona, Bilbao, Burgos, Caceres, Campo de Gibraltar, Castellon de la Plana, Ceuta, Ciudad Real, Cordoba, Cuenca, El Hierro, Ferrol, Fuerteventura, Gran Canaria, Granada, Huelva, Huesca, Ibiza, Jaén - Úbeda, La Gomera, La Palma, Lanzarote, León, Lleida, Logroño, Lugo, Madrid, Malaga, Mallorca - Majorca, Melilla, Menorca, Merida, Murcia, Ourense, Palencia, Pamplona, Salamanca, San Sebastian, Santander, Santiago de Compostela, Segovia, Seville, Soria, Tarragona - Reus, Tenerife, Toledo, Valencia, Valladolid, Vigo, Vitoria-Gasteiz, Zaragoza - Saragossa
Sweden	Goteborg/Gothenburg/Jönköping, Malmö kommun - Malmö, Norrköping och Linköping, Stockholm, Sundsvall

COUNTRY/REGION	CITY (METRO AREA)
Switzerland	Basel, Geneva, Yverdon-les-Bains, Zurich
Turkey	Adana-Mersin, Ankara, Antalya, Balıkesir, Bilecik, Bolu, Bursa, Çorum, Denizli, Duzce, Edirne, Elazig, Eskisehir, Istanbul, Izmir-Aydin, Kahramanmaraş, Kayseri, Konya, Malatya, Muğla, Samsun, Şanlıurfa, Trabzon
United Kingdom	East Anglia, East Midlands, London and South East, North East, North West, Northern Ireland, Scotland, South West, Wales, West Midlands, Yorkshire
Ukraine	Kharkiv, Zhytomyr, Kyiv, Lviv, Chernivtsi

Middle East and Africa

COUNTRY/REGION	CITY (METRO AREA)
Bahrain	Bahrain*
Burkina Faso	Ouagadougou
Congo	Kinshasa
Egypt	Cairo
Israel	Israel*
Kenya	Nairobi
Madagascar	Antananarivo
Morocco	Casablanca, Essaouira, Khouribga, Tétouan
Qatar	Doha
Saudi Arabia	Thuwal
Senegal	Dakar
South Africa	Cape Town
Tunisia	Kairouan
United Arab Emirates	Abu Dhabi, Dubai

Next steps

Learn how to request transit data using Mobility services (Preview):

[How to request transit data](#)

Learn how to request real-time data using Mobility services (Preview):

[How to request real-time data](#)

Explore the Azure Maps Mobility services (Preview) API documentation

[Mobility services API documentation](#)

Azure Maps Weather services coverage

4/9/2021 • 5 minutes to read • [Edit Online](#)

This article provides coverage information for Azure Maps [Weather services](#). Azure Maps Weather data services returns details such as radar tiles, current weather conditions, weather forecasts, and the weather along a route.

Azure Maps doesn't have the same level of information and accuracy for all countries and regions.

The following table provides information about what kind of weather information you can request from each country/region.

SYMBOL	MEANING
*	Covers Current Conditions, Hourly Forecast, Quarter-day Forecast, Daily Forecast, Weather Along Route and Daily Indices.

Americas

COUNTRY/REGION	SATELLITE TILES	MINUTE FORECAST, RADAR TILES	SEVERE WEATHER ALERTS	OTHER*
Anguilla	✓			✓
Antarctica	✓			✓
Antigua and Barbuda	✓			✓
Argentina	✓			✓
Aruba	✓			✓
Bahamas	✓			✓
Barbados	✓			✓
Belize	✓			✓
Bermuda	✓			✓
Bolivia	✓			✓
Bonaire	✓			✓
Brazil	✓		✓	✓
British Virgin Islands	✓			✓
Canada	✓	✓	✓	✓

Country/Region	Satellite Tiles	Minute Forecast, Radar Tiles	Severe Weather Alerts	Other*
Cayman Islands	✓			✓
Chile	✓			✓
Colombia	✓			✓
Costa Rica	✓			✓
Cuba	✓			✓
Curaçao	✓			✓
Dominica	✓			✓
Dominican Republic	✓			✓
Ecuador	✓			✓
El Salvador	✓			✓
Falkland Islands	✓			✓
French Guiana	✓			✓
Greenland	✓			✓
Grenada	✓			✓
Guadeloupe	✓			✓
Guatemala	✓			✓
Guyana	✓			✓
Haiti	✓			✓
Honduras	✓			✓
Jamaica	✓			✓
Martinique	✓			✓
Mexico	✓			✓
Montserrat	✓			✓
Nicaragua	✓			✓
Panama	✓			✓

Country/Region	Satellite Tiles	Minute Forecast, Radar Tiles	Severe Weather Alerts	Other*
Paraguay	✓			✓
Peru	✓			✓
Puerto Rico	✓		✓	✓
Saint Barthélemy	✓			✓
Saint Kitts and Nevis	✓			✓
Saint Lucia	✓			✓
Saint Martin	✓			✓
Saint Pierre and Miquelon	✓			✓
Saint Vincent and the Grenadines	✓			✓
Sint Eustatius	✓			✓
Sint Maarten	✓			✓
South Georgia and South Sandwich Islands	✓			✓
Suriname	✓			✓
Trinidad and Tobago	✓			✓
Turks and Caicos Islands	✓			✓
U.S. Outlying Islands	✓			✓
U.S. Virgin Islands	✓		✓	✓
United States	✓	✓	✓	✓
Uruguay	✓			✓
Venezuela	✓			✓

Middle East and Africa

Country/Region	Satellite Tiles	Minute Forecast, Radar Tiles	Severe Weather Alerts	Other*
Algeria	✓			✓
Angola	✓			✓
Bahrain	✓			✓
Benin	✓			✓
Botswana	✓			✓
Bouvet Island	✓			✓
Burkina Faso	✓			✓
Burundi	✓			✓
Cameroon	✓			✓
Cabo Verde	✓			✓
Central African Republic	✓			✓
Chad	✓			✓
Comoros	✓			✓
Congo (DRC)	✓			✓
Côte d'Ivoire	✓			✓
Djibouti	✓			✓
Egypt	✓			✓
Equatorial Guinea	✓			✓
Eritrea	✓			✓
eSwatini	✓			✓
Ethiopia	✓			✓
French Southern Territories	✓			✓
Gabon	✓			✓
Gambia	✓			✓

Country/Region	Satellite Tiles	Minute Forecast, Radar Tiles	Severe Weather Alerts	Other*
Ghana	✓			✓
Guinea	✓			✓
Guinea-Bissau	✓			✓
Iran	✓			✓
Iraq	✓			✓
Israel	✓		✓	✓
Jordan	✓			✓
Kenya	✓			✓
Kuwait	✓			✓
Lebanon	✓			✓
Lesotho	✓			✓
Liberia	✓			✓
Libya	✓			✓
Madagascar	✓			✓
Malawi	✓			✓
Mali	✓			✓
Mauritania	✓			✓
Mauritius	✓			✓
Mayotte	✓			✓
Morocco	✓			✓
Mozambique	✓			✓
Namibia	✓			✓
Niger	✓			✓
Nigeria	✓			✓
Oman	✓			✓

Country/Region	Satellite Tiles	Minute Forecast, Radar Tiles	Severe Weather Alerts	Other*
Palestinian Authority	✓			✓
Qatar	✓			✓
Réunion	✓			✓
Rwanda	✓			✓
St Helena, Ascension, Tristan da Cunha	✓			✓
São Tomé and Príncipe	✓			✓
Saudi Arabia	✓			✓
Senegal	✓			✓
Seychelles	✓			✓
Sierra Leone	✓			✓
Somalia	✓			✓
South Africa	✓			✓
South Sudan	✓			✓
Sudan	✓			✓
Syria	✓			✓
Tanzania	✓			✓
Togo	✓			✓
Tunisia	✓			✓
Uganda	✓			✓
United Arab Emirates	✓			✓
Yemen	✓			✓
Zambia	✓			✓
Zimbabwe	✓			✓

Asia Pacific

Country/Region	Satellite Tiles	Minute Forecast, Radar Tiles	Severe Weather Alerts	Other*
Afghanistan	✓			✓
American Samoa	✓		✓	✓
Australia	✓	✓	✓	✓
Bangladesh	✓			✓
Bhutan	✓			✓
British Indian Ocean Territory	✓			✓
Brunei	✓			✓
Cambodia	✓			✓
China	✓	✓	✓	✓
Christmas Island	✓			✓
Cocos (Keeling) Islands	✓			✓
Cook Islands	✓			✓
Fiji	✓			✓
French Polynesia	✓			✓
Guam	✓		✓	✓
Heard Island and McDonald Islands	✓			✓
Hong Kong SAR	✓			✓
India	✓			✓
Indonesia	✓			✓
Japan	✓	✓	✓	✓
Kazakhstan	✓			✓
Kiribati	✓			✓
Korea	✓	✓	✓	✓
Kyrgyzstan	✓			✓

Country/Region	Satellite Tiles	Minute Forecast, Radar Tiles	Severe Weather Alerts	Other*
Laos	✓			✓
Macao SAR	✓			✓
Malaysia	✓			✓
Maldives	✓			✓
Marshall Islands	✓		✓	✓
Micronesia	✓		✓	✓
Mongolia	✓			✓
Myanmar	✓			✓
Nauru	✓			✓
Nepal	✓			✓
New Caledonia	✓			✓
New Zealand	✓		✓	✓
Niue	✓			✓
Norfolk Island	✓			✓
North Korea	✓			✓
Northern Mariana Islands	✓		✓	✓
Pakistan	✓			✓
Palau	✓		✓	✓
Papua New Guinea	✓			✓
Philippines	✓		✓	✓
Pitcairn Islands	✓			✓
Samoa	✓			✓
Singapore	✓			✓
Solomon Islands	✓			✓
Sri Lanka	✓			✓

Country/Region	Satellite Tiles	Minute Forecast, Radar Tiles	Severe Weather Alerts	Other*
Taiwan	✓			✓
Tajikistan	✓			✓
Thailand	✓			✓
Timor-Leste	✓			✓
Tokelau	✓			✓
Tonga	✓			✓
Turkmenistan	✓			✓
Tuvalu	✓			✓
Uzbekistan	✓			✓
Vanuatu	✓			✓
Vietnam	✓			✓
Wallis and Futuna	✓			✓

Europe

Country/Region	Satellite Tiles	Minute Forecast, Radar Tiles	Severe Weather Alerts	Other*
Albania	✓			✓
Andorra	✓		✓	✓
Armenia	✓			✓
Austria	✓	✓	✓	✓
Azerbaijan	✓			✓
Belarus	✓			✓
Belgium	✓	✓	✓	✓
Bosnia and Herzegovina	✓	✓	✓	✓
Bulgaria	✓		✓	✓
Croatia	✓	✓	✓	✓

Country/Region	Satellite Tiles	Minute Forecast, Radar Tiles	Severe Weather Alerts	Other*
Cyprus	✓		✓	✓
Czechia	✓	✓	✓	✓
Denmark	✓	✓	✓	✓
Estonia	✓	✓	✓	✓
Faroe Islands	✓			✓
Finland	✓	✓	✓	✓
France	✓	✓	✓	✓
Georgia	✓			✓
Germany	✓	✓	✓	✓
Gibraltar	✓	✓		✓
Greece	✓		✓	✓
Guernsey	✓			✓
Hungary	✓	✓	✓	✓
Iceland	✓		✓	✓
Ireland	✓	✓	✓	✓
Italy	✓		✓	✓
Isle of Man	✓			✓
Jan Mayen	✓			✓
Jersey	✓			✓
Kosovo	✓		✓	✓
Latvia	✓		✓	✓
Liechtenstein	✓	✓	✓	✓
Lithuania	✓		✓	✓
Luxembourg	✓	✓	✓	✓
North Macedonia	✓		✓	✓

Country/Region	Satellite Tiles	Minute Forecast, Radar Tiles	Severe Weather Alerts	Other*
Malta	✓		✓	✓
Moldova	✓	✓	✓	✓
Monaco	✓	✓	✓	✓
Montenegro	✓	✓	✓	✓
Netherlands	✓	✓	✓	✓
Norway	✓	✓	✓	✓
Poland	✓	✓	✓	✓
Portugal	✓	✓	✓	✓
Romania	✓	✓	✓	✓
Russia	✓		✓	✓
San Marino	✓		✓	✓
Serbia	✓	✓	✓	✓
Slovakia	✓	✓	✓	✓
Slovenia	✓	✓	✓	✓
Spain	✓	✓	✓	✓
Svalbard	✓			✓
Sweden	✓	✓	✓	✓
Switzerland	✓	✓	✓	✓
Turkey	✓			✓
Ukraine	✓			✓
United Kingdom	✓	✓	✓	✓
Vatican City	✓		✓	✓

Localization support in Azure Maps

3/5/2021 • 4 minutes to read • [Edit Online](#)

Azure Maps supports various languages and views based on country/region. This article provides the supported languages and views to help guide your Azure Maps implementation.

Azure Maps supported languages

Azure Maps have been localized in variety languages across its services. The following table provides the supported language codes for each service.

ID	NAME	MAPS	SEARCH	ROUTING	WEATHER	TRAFFIC INCIDENTS	JS MAP CONTROL
af-ZA	Afrikaans		✓	✓			
ar-SA	Arabic	✓	✓	✓	✓	✓	✓
bn-BD	Bangla (Bangladesh)				✓		
bn-IN	Bangla (India)				✓		
bs-BA	Bosnian				✓		
eu-ES	Basque		✓				
bg-BG	Bulgarian	✓	✓	✓	✓		✓
ca-ES	Catalan		✓		✓		
zh-HanS	Chinese (Simplified)		zh-CN		zh-CN		
zh-HanT	Chinese (Hong Kong SAR)				zh-HK		
zh-HanT	Chinese (Taiwan)	zh-TW	zh-TW	zh-TW	zh-TW		zh-TW
hr-HR	Croatian		✓		✓		
cs-CZ	Czech	✓	✓	✓	✓	✓	✓
da-DK	Danish	✓	✓	✓	✓	✓	✓
nl-BE	Dutch (Belgium)		✓		✓		

ID	NAME	MAPS	SEARCH	ROUTING	WEATHER	TRAFFIC INCIDENTS	JS MAP CONTROL
nl-NL	Dutch (Netherlands)	✓	✓	✓	✓	✓	✓
en-AU	English (Australia)	✓	✓	✓	✓	✓	✓
en-NZ	English (New Zealand)	✓	✓	✓	✓	✓	✓
en-GB	English (Great Britain)	✓	✓	✓	✓	✓	✓
en-US	English (USA)	✓	✓	✓	✓	✓	✓
et-EE	Estonian		✓		✓	✓	
fil-PH	Filipino				✓		
fi-FI	Finnish	✓	✓	✓	✓	✓	✓
fr-FR	French	✓	✓	✓	✓	✓	✓
fr-CA	French (Canada)		✓		✓		
gl-ES	Galician		✓				
de-DE	German	✓	✓	✓	✓	✓	✓
el-GR	Greek	✓	✓	✓	✓	✓	✓
gu-IN	Gujarati				✓		
he-IL	Hebrew		✓		✓	✓	
hi-IN	Hindi				✓		
hu-HU	Hungarian	✓	✓	✓	✓	✓	✓
is-IS	Icelandic				✓		
id-ID	Indonesian	✓	✓	✓	✓	✓	✓
it-IT	Italian	✓	✓	✓	✓	✓	✓
ja-JP	Japanese				✓		

ID	NAME	MAPS	SEARCH	ROUTING	WEATHER	TRAFFIC INCIDENTS	JS MAP CONTROL
kn-IN	Kannada				✓		
kk-KZ	Kazakh		✓		✓		
ko-KR	Korean	✓		✓	✓		✓
es-419	Latin American Spanish		✓				
lv-LV	Latvian		✓		✓	✓	
lt-LT	Lithuanian	✓	✓	✓	✓	✓	✓
mk-MK	Macedonian				✓		
ms-MY	Malay (Latin)	✓	✓	✓	✓		✓
mr-IN	Marathi				✓		
nb-NO	Norwegian Bokmål	✓	✓	✓	✓	✓	✓
NGT	Neutral Ground Truth - Official languages for all regions in local scripts if available	✓					✓
NGT-Latn	Neutral Ground Truth - Latin exonyms. Latin script will be used if available	✓					✓
pl-PL	Polish	✓	✓	✓	✓	✓	✓
pt-BR	Portuguese (Brazil)	✓	✓	✓	✓		✓
pt-PT	Portuguese (Portugal)	✓	✓	✓	✓	✓	✓
pa-IN	Punjabi				✓		

ID	NAME	MAPS	SEARCH	ROUTING	WEATHER	TRAFFIC INCIDENTS	JS MAP CONTROL
ro-RO	Romanian		✓		✓	✓	
ru-RU	Russian	✓	✓	✓	✓	✓	✓
sr-Cyrl-RS	Serbian (Cyrillic)		sr-RS		sr-RS		
sr-Latn-RS	Serbian (Latin)				sr-latn		
sk-SK	Slovak	✓	✓	✓	✓	✓	✓
sl-SL	Slovenian	✓	✓	✓	✓		✓
es-ES	Spanish	✓	✓	✓	✓	✓	✓
es-MX	Spanish (Mexico)	✓		✓	✓		✓
sv-SE	Swedish	✓	✓	✓	✓	✓	✓
ta-IN	Tamil (India)				✓		
te-IN	Telugu (India)				✓		
th-TH	Thai	✓	✓	✓	✓	✓	✓
tr-TR	Turkish	✓	✓	✓	✓	✓	✓
uk-UA	Ukrainian		✓		✓		
ur-PK	Urdu				✓		
uz-Latn-UZ	Uzbek				✓		
vi-VN	Vietnamese		✓		✓		

Azure Maps supported views

NOTE

On August 1, 2019, Azure Maps was released in the following countries/regions:

- Argentina
- India
- Morocco
- Pakistan

After August 1, 2019, the **View** parameter will define the returned map content for the new regions/countries listed above. Azure Maps **View** parameter (also referred to as "user region parameter") is a two letter ISO-3166 Country Code that will show the correct maps for that country/region specifying which set of geopolitically disputed content is returned via Azure Maps services, including borders and labels displayed on the map.

Make sure you set up the **View** parameter as required for the REST APIs and the SDKs, which your services are using.

Rest APIs

Ensure that you have set up the **View** parameter as required. **View** parameter specifies which set of geopolitically disputed content is returned via Azure Maps services.

Affected Azure Maps REST Services:

- Get Map Tile
- Get Map Image
- Get Search Fuzzy
- Get Search POI
- Get Search POI Category
- Get Search Nearby
- Get Search Address
- Get Search Address Structured
- Get Search Address Reverse
- Get Search Address Reverse Cross Street
- Post Search Inside Geometry
- Post Search Address Batch
- Post Search Address Reverse Batch
- Post Search Along Route
- Post Search Fuzzy Batch

SDKs

Ensure that you have set up the **View** parameter as required, and you have the latest version of Web SDK and Android SDK. Affected SDKs:

- Azure Maps Web SDK
- Azure Maps Android SDK

By default, the **View** parameter is set to **Unified**, even if you haven't defined it in the request. Determine the location of your users. Then, set the **View** parameter correctly for that location. Alternatively, you can set 'View=Auto', which will return the map data based on the IP address of the request. The **View** parameter in Azure Maps must be used in compliance with applicable laws, including those laws about mapping of the country/region where maps, images, and other data and third-party content that you're authorized to access via Azure Maps is made available.

The following table provides supported views.

VIEW	DESCRIPTION	MAPS	SEARCH	JS MAP CONTROL
AE	United Arab Emirates (Arabic View)	✓		✓
AR	Argentina (Argentinian View)	✓	✓	✓
BH	Bahrain (Arabic View)	✓		✓
IN	India (Indian View)	✓	✓	✓
IQ	Iraq (Arabic View)	✓		✓
JO	Jordan (Arabic View)	✓		✓
KW	Kuwait (Arabic View)	✓		✓
LB	Lebanon (Arabic View)	✓		✓
MA	Morocco (Moroccan View)	✓	✓	✓
OM	Oman (Arabic View)	✓		✓
PK	Pakistan (Pakistani View)	✓	✓	✓
PS	Palestinian Authority (Arabic View)	✓		✓
QA	Qatar (Arabic View)	✓		✓
SA	Saudi Arabia (Arabic View)	✓		✓
SY	Syria (Arabic View)	✓		✓
YE	Yemen (Arabic View)	✓		✓
Auto	Return the map data based on the IP address of the request.	✓	✓	✓
Unified	Unified View (Others)	✓	✓	✓

Azure Maps supported built-in map styles

4/30/2021 • 2 minutes to read • [Edit Online](#)

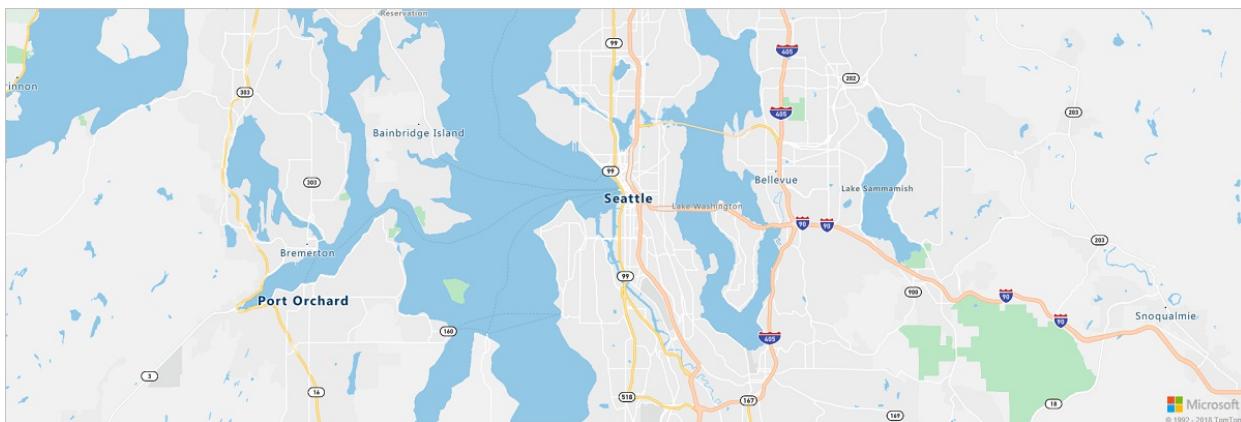
Azure Maps supports several different built-in map styles as described below.

IMPORTANT

The procedure in this section requires an Azure Maps account in Gen 1 or Gen 2 pricing tier. For more information on pricing tiers, see [Choose the right pricing tier in Azure Maps](#).

road

A **road** map is a standard map that displays roads. It also displays natural and artificial features, and the labels for those features.



Applicable APIs:

- [Map image](#)
- [Map tile](#)
- [Web SDK map control](#)
- [Android map control](#)
- [Power BI visual](#)

blank and blank_accessible

The **blank** and **blank_accessible** map styles provide a blank canvas for visualizing data. The **blank_accessible** style will continue to provide screen reader updates with map's location details, even though the base map isn't displayed.

NOTE

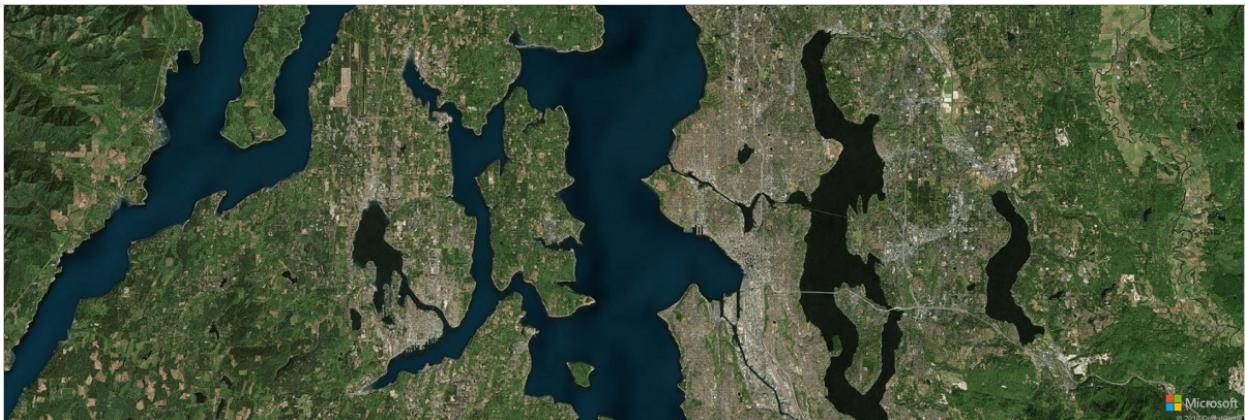
In the Web SDK, you can change the background color of the map by setting the CSS `background-color` style of map DIV element.

Applicable APIs:

- [Web SDK map control](#)

satellite

The **satellite** style is a combination of satellite and aerial imagery.

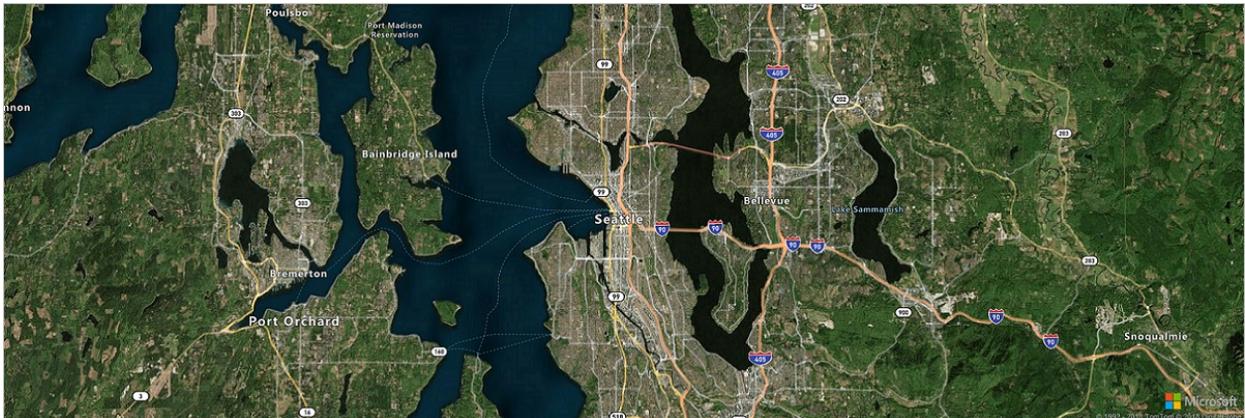


Applicable APIs:

- [Satellite tile](#)
- Web SDK map control
- Android map control
- Power BI visual

satellite_road_labels

This map style is a hybrid of roads and labels overlaid on top of satellite and aerial imagery.

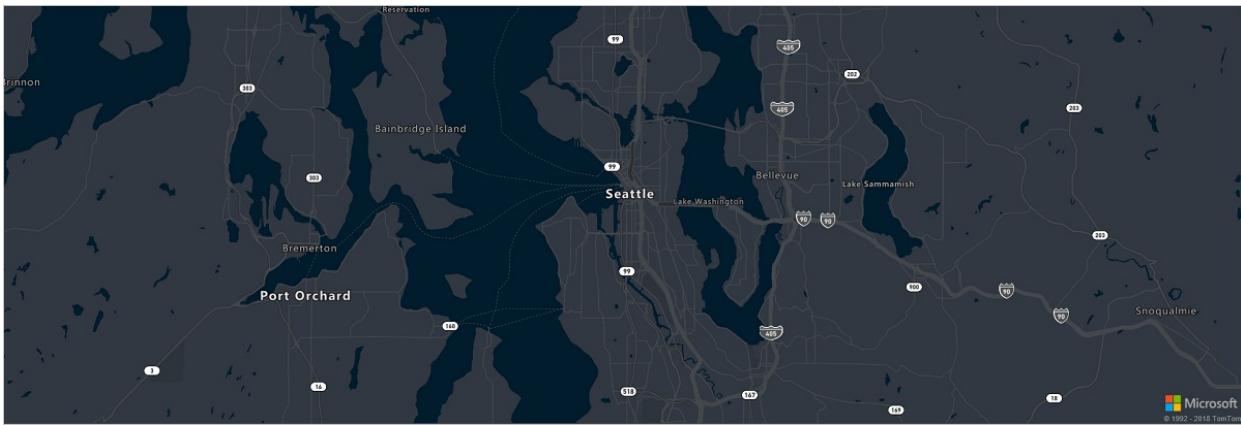


Applicable APIs:

- Web SDK map control
- Android map control
- Power BI visual

grayscale_dark

grayscale dark is a dark version of the road map style.



Applicable APIs:

- [Map image](#)
- [Map tile](#)
- Web SDK map control
- Android map control
- Power BI visual

grayscale_light

grayscale light is a light version of the road map style.

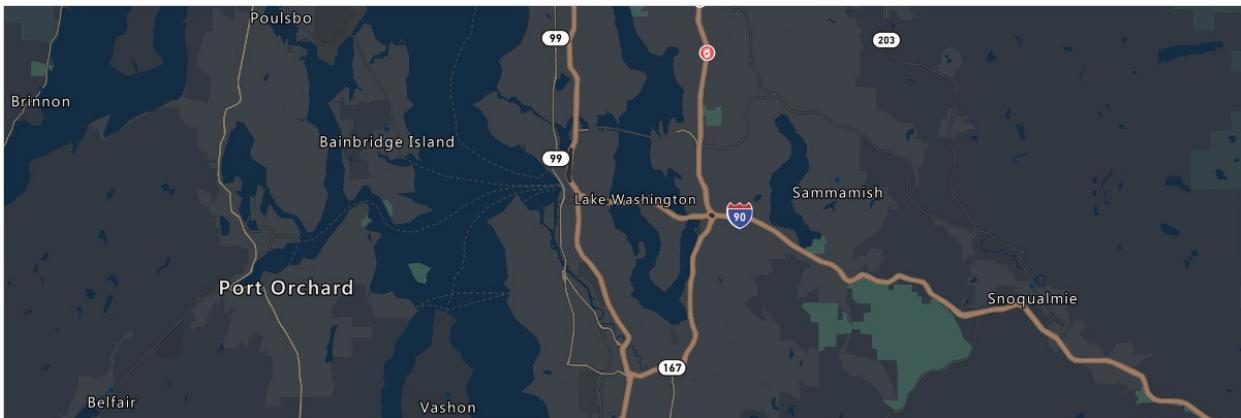


Applicable APIs:

- Web SDK map control
- Android map control
- Power BI visual

night

night is a dark version of the road map style with colored roads and symbols.



Applicable APIs:

- Web SDK map control
- Android map control
- Power BI visual

road_shaded_relief

road shaded relief is an Azure Maps main style completed with contours of the Earth.

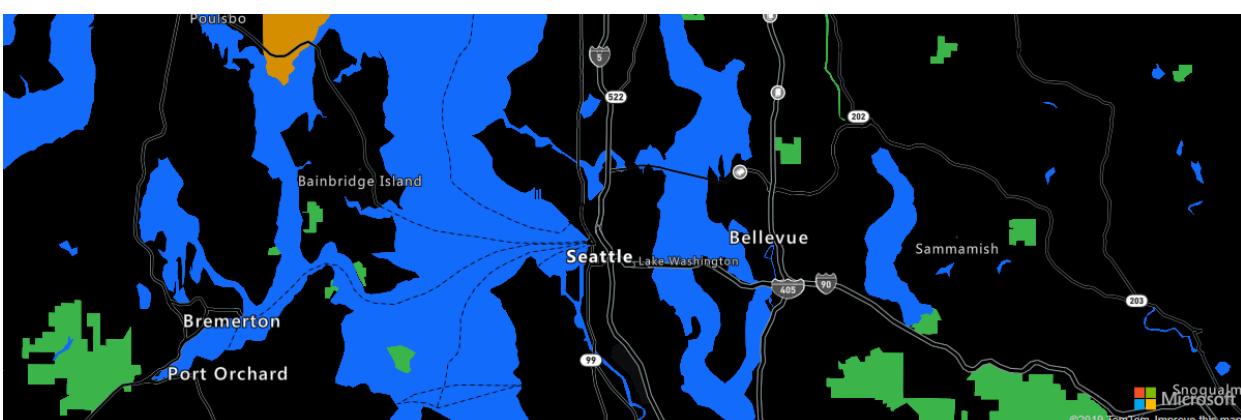


Applicable APIs:

- Map tile
- Web SDK map control
- Android map control
- Power BI visual

high_contrast_dark

high_contrast_dark is a dark map style with a higher contrast than the other styles.



Applicable APIs:

- Web SDK map control
- Power BI visual

Next steps

Learn about how to set a map style in Azure Maps:

[Choose a map style](#)

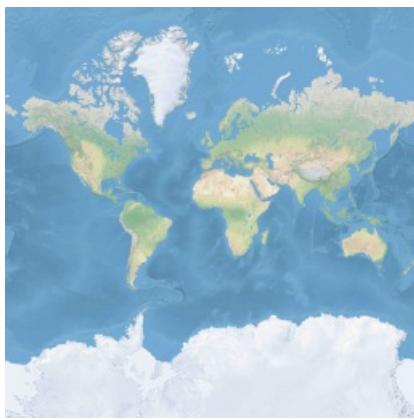
Zoom levels and tile grid

11/2/2020 • 28 minutes to read • [Edit Online](#)

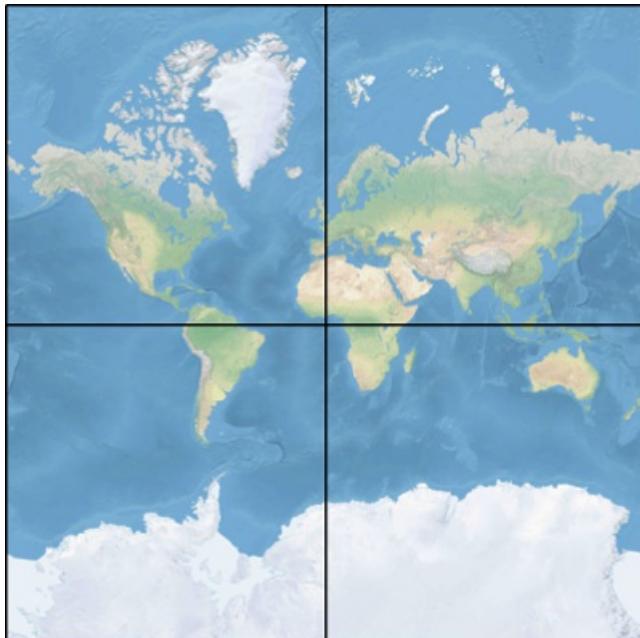
Azure Maps use the Spherical Mercator projection coordinate system (EPSG: 3857). A projection is the mathematical model used to transform the spherical globe into a flat map. The Spherical Mercator projection stretches the map at the poles to create a square map. This projection significantly distorts the scale and area of the map but has two important properties that outweigh this distortion:

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

To optimize the performance of map retrieval and display, the map is divided into square tiles. The Azure Maps SDK's use tiles that have a size of 512 x 512 pixels for road maps, and smaller 256 x 256 pixels for satellite imagery. Azure Maps provides raster and vector tiles for 23 zoom levels, numbered 0 through 22. At zoom level 0, the entire world fits on a single tile:



Zoom level 1 uses four tiles to render the world: a 2 x 2 square



Each additional zoom level quad-divides the tiles of the previous one, creating a grid of $2^{\text{zoom}} \times 2^{\text{zoom}}$. Zoom

level 22 is a grid $2^{22} \times 2^{22}$, or 4,194,304 x 4,194,304 tiles (17,592,186,044,416 tiles in total).

The Azure Maps interactive map controls for web and Android support 25 zoom levels, numbered 0 through 24. Although road data will only be available at the zoom levels in when the tiles are available.

The following table provides the full list of values for zoom levels where the tile size is 512 pixels square at latitude 0:

ZOOM LEVEL	METERS/PIXEL	METERS/TILE SIDE
0	156543	40075017
1	78271.5	20037508
2	39135.8	10018754
3	19567.88	5009377.1
4	9783.94	2504688.5
5	4891.97	1252344.3
6	2445.98	626172.1
7	1222.99	313086.1
8	611.5	156543
9	305.75	78271.5
10	152.87	39135.8
11	76.44	19567.9
12	38.219	9783.94
13	19.109	4891.97
14	9.555	2445.98
15	4.777	1222.99
16	2.3887	611.496
17	1.1943	305.748
18	0.5972	152.874
19	0.14929	76.437
20	0.14929	38.2185
21	0.074646	19.10926

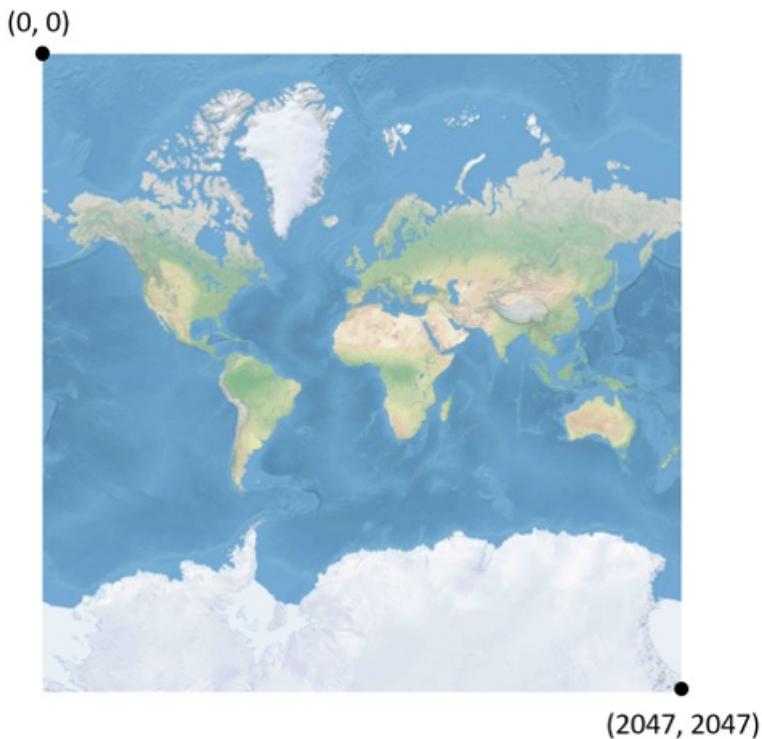
ZOOM LEVEL	METERS/PIXEL	METERS/TILE SIDE
22	0.037323	9.55463
23	0.0186615	4.777315
24	0.00933075	2.3886575

Pixel coordinates

Having chosen the projection and scale to use at each zoom level, we can convert geographic coordinates into pixel coordinates. The full pixel width and height of a map image of the world for a particular zoom level is calculated as:

```
var mapWidth = tileSize * Math.pow(2, zoom);
var mapHeight = mapWidth;
```

Since the map width and height is different at each zoom level, so are the pixel coordinates. The pixel at the upper-left corner of the map always has pixel coordinates (0, 0). The pixel at the lower-right corner of the map has pixel coordinates ($width - 1, height - 1$), or referring to the equations in the previous section, ($tileSize * 2^{zoom} - 1, tileSize * 2^{zoom} - 1$). For example, when using 512 square tiles at level 2, the pixel coordinates range from (0, 0) to (2047, 2047), like this:



Given latitude and longitude in degrees, and the level of detail, the pixel XY coordinates is calculated as follows:

```
var sinLatitude = Math.sin(latitude * Math.PI/180);
var pixelX = ((longitude + 180) / 360) * tileSize * Math.pow(2, zoom);
var pixelY = (0.5 - Math.log((1 + sinLatitude) / (1 - sinLatitude)) / (4 * Math.PI)) * tileSize *
Math.pow(2, zoom);
```

The latitude and longitude values are assumed to be on the WGS 84 datum. Even though Azure Maps uses a spherical projection, it's important to convert all geographic coordinates into a common datum. WGS 84 is the selected datum. The longitude value is assumed to range from -180 degrees to +180 degrees, and the latitude value must be clipped to range from -85.05112878 to 85.05112878. Adhering to these values avoids a singularity at the poles, and it ensures that the projected map is a squared shape.

Tile coordinates

To optimize the performance of map retrieval and display, the rendered map is cut into tiles. The number of pixels and the number of tiles differ at each zoom level:

```
var numberOfTilesWide = Math.pow(2, zoom);  
  
var numberOfTilesHigh = numberOfTilesWide;
```

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

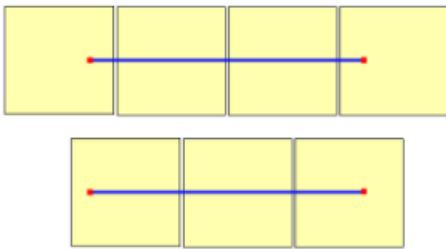
(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)	(7,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)	(7,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)	(7,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)	(7,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)	(7,4)
(0,5)	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)	(7,5)
(0,6)	(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(6,6)	(7,6)
(0,7)	(1,7)	(2,7)	(3,7)	(4,7)	(5,7)	(6,7)	(7,7)

Given a pair of pixel XY coordinates, you can easily determine the tile XY coordinates of the tile containing that pixel:

```
var tileX = Math.floor(pixelX / tileSize);  
  
var tileY = Math.floor(pixelY / tileSize);
```

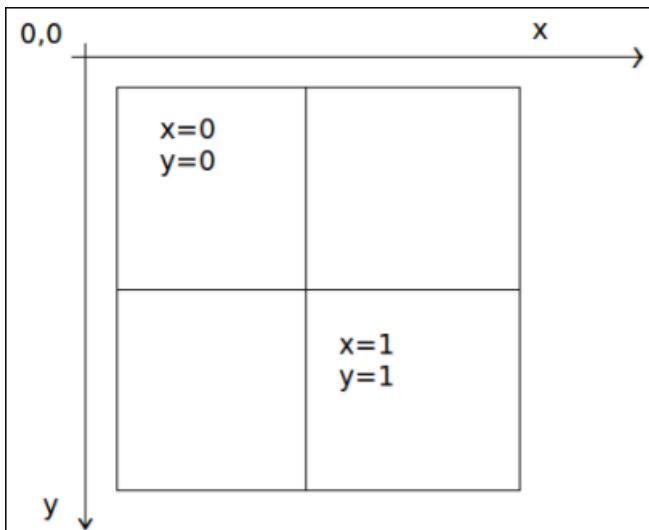
Tiles are called by zoom level. The x and y coordinates correspond to the tile's position on the grid for that zoom level.

When determining which zoom level to use, remember each location is in a fixed position on its tile. As a result, the number of tiles needed to display a given expanse of territory is dependent on the specific placement of zoom grid on the world map. For instance, if there are two points 900 meters apart, it *may* only take three tiles to display a route between them at zoom level 17. However, if the western point is on the right of its tile, and the eastern point on the left of its tile, it may take four tiles:



Once the zoom level is determined, the x and y values can be calculated. The top-left tile in each zoom grid is $x=0, y=0$; the bottom-right tile is at $x=2^{\text{zoom}-1}, y=2^{\text{zoom}-1}$.

Here is the zoom grid for zoom level 1:



Quadkey indices

Some mapping platforms use a `quadkey` indexing naming convention that combines the tile ZY coordinates into a one-dimension string called `quadtree` keys or `quadkeys` for short. Each `quadkey` uniquely identifies a single tile at a particular level of detail, and it can be used as a key in common database B-tree indexes. The Azure Maps SDKs support the overlaying of tile layers that use `quadkey` naming convention in addition to other naming conventions as documented in the [Add a tile layer](#) document.

NOTE

The `quadkeys` naming convention only works for zoom levels of one or greater. The Azure Maps SDK's support zoom level 0 which is a single map tile for the whole world.

To convert tile coordinates into a `quadkey`, the bits of the Y and X coordinates are interleaved, and the result is interpreted as a base-4 number (with leading zeros maintained) and converted into a string. For instance, given tile XY coordinates of (3, 5) at level 3, the `quadkey` is determined as follows:

```

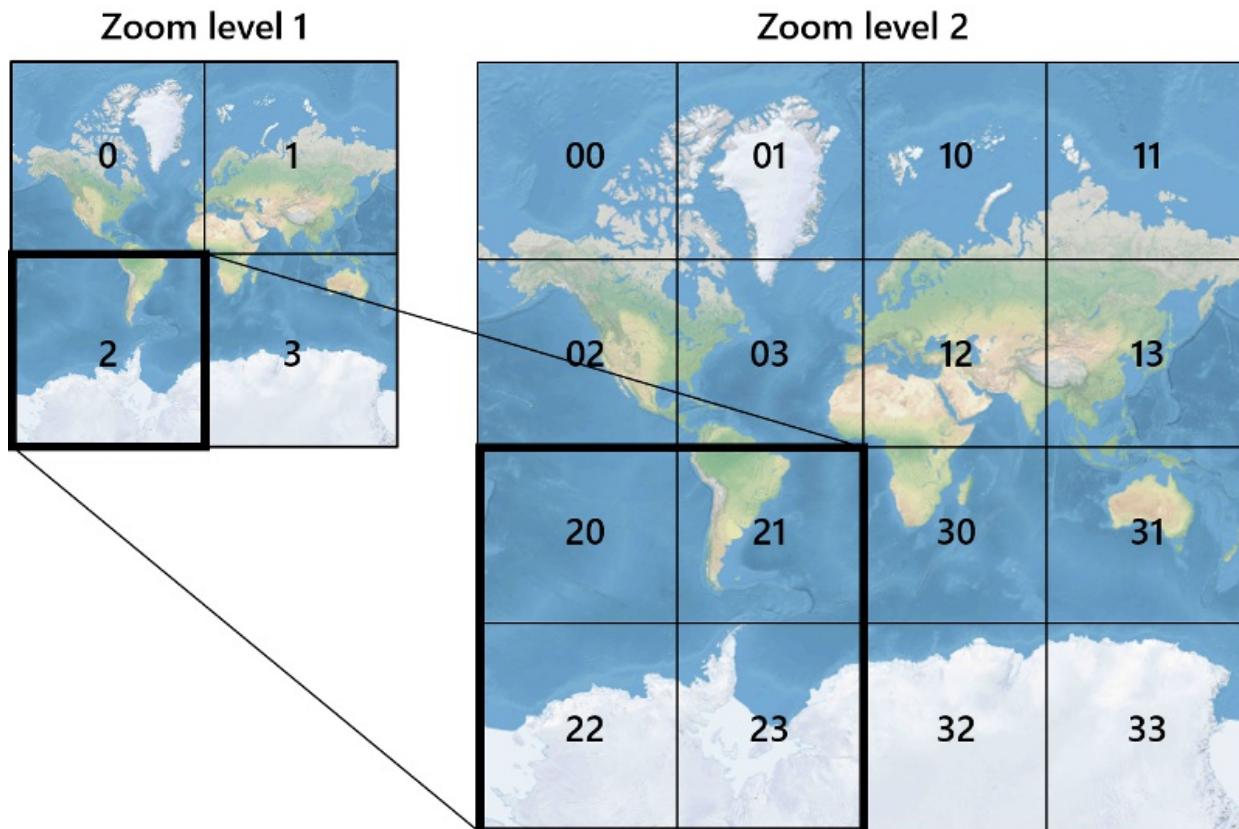
tileX = 3 = 011 (base 2)

tileY = 5 = 101 (base 2)

quadkey = 100111 (base 2) = 213 (base 4) = "213"

```

`quadkeys` have several interesting properties. First, the length of a `quadkey` (the number of digits) equals 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:



Finally, `quadkeys` provide a one-dimensional index key that usually preserves the proximity of tiles in XY space. In other words, two tiles that have nearby XY coordinates usually have `quadkeys` that are relatively close together. This is important for optimizing database performance, because neighboring tiles are often requested in groups, and it's desirable to keep those tiles on the same disk blocks, in order to minimize the number of disk reads.

Tile math source code

The following sample code illustrates how to implement the functions described in this document. These functions can be easily translated into other programming languages as needed.

- [C#](#)
- [TypeScript](#)

```

using System;
using System.Text;

namespace AzureMaps
{
    /// <summary>
    /// Tile System math for the Spherical Mercator projection coordinate system (EPSG:3857)
    /// 
```

```

/// </summary>
public static class TileMath
{
    //Earth radius in meters.
    private const double EarthRadius = 6378137;

    private const double MinLatitude = -85.05112878;
    private const double MaxLatitude = 85.05112878;
    private const double MinLongitude = -180;
    private const double MaxLongitude = 180;

    /// <summary>
    /// Clips a number to the specified minimum and maximum values.
    /// </summary>
    /// <param name="n">The number to clip.</param>
    /// <param name="minValue">Minimum allowable value.</param>
    /// <param name="maxValue">Maximum allowable value.</param>
    /// <returns>The clipped value.</returns>
    private static double Clip(double n, double minValue, double maxValue)
    {
        return Math.Min(Math.Max(n, minValue), maxValue);
    }

    /// <summary>
    /// Calculates width and height of the map in pixels at a specific zoom level from -180 degrees to
    180 degrees.
    /// </summary>
    /// <param name="zoom">Zoom Level to calculate width at</param>
    /// <param name="tileSize">The size of the tiles in the tile pyramid.</param>
    /// <returns>Width and height of the map in pixels</returns>
    public static double MapSize(double zoom, int tileSize)
    {
        return Math.Ceiling(tileSize * Math.Pow(2, zoom));
    }

    /// <summary>
    /// Calculates the Ground resolution at a specific degree of latitude in meters per pixel.
    /// </summary>
    /// <param name="latitude">Degree of latitude to calculate resolution at</param>
    /// <param name="zoom">Zoom level to calculate resolution at</param>
    /// <param name="tileSize">The size of the tiles in the tile pyramid.</param>
    /// <returns>Ground resolution in meters per pixels</returns>
    public static double GroundResolution(double latitude, double zoom, int tileSize)
    {
        latitude = Clip(latitude, MinLatitude, MaxLatitude);
        return Math.Cos(latitude * Math.PI / 180) * 2 * Math.PI * EarthRadius / MapSize(zoom, tileSize);
    }

    /// <summary>
    /// Determines the map scale at a specified latitude, level of detail, and screen resolution.
    /// </summary>
    /// <param name="latitude">Latitude (in degrees) at which to measure the map scale.</param>
    /// <param name="zoom">Level of detail, from 1 (lowest detail) to 23 (highest detail).</param>
    /// <param name="screenDpi">Resolution of the screen, in dots per inch.</param>
    /// <param name="tileSize">The size of the tiles in the tile pyramid.</param>
    /// <returns>The map scale, expressed as the denominator N of the ratio 1 : N.</returns>
    public static double MapScale(double latitude, double zoom, int screenDpi, int tileSize)
    {
        return GroundResolution(latitude, zoom, tileSize) * screenDpi / 0.0254;
    }

    /// <summary>
    /// Global Converts a Pixel coordinate into a geospatial coordinate at a specified zoom level.
    /// Global Pixel coordinates are relative to the top left corner of the map (90, -180)
    /// </summary>
    /// <param name="pixel">Pixel coordinates in the format of [x, y].</param>
    /// <param name="zoom">Zoom level</param>
    /// <param name="tileSize">The size of the tiles in the tile pyramid.</param>
    /// <returns>A position value in the format [longitude, latitude].</returns>

```

```

public static double[] GlobalPixelToPosition(double[] pixel, double zoom, int tileSize)
{
    var mapSize = MapSize(zoom, tileSize);

    var x = (Clip(pixel[0], 0, mapSize - 1) / mapSize) - 0.5;
    var y = 0.5 - (Clip(pixel[1], 0, mapSize - 1) / mapSize);

    return new double[] {
        360 * x,          //Longitude
        90 - 360 * Math.Atan(Math.Exp(-y * 2 * Math.PI)) / Math.PI //Latitude
    };
}

/// <summary>
/// Converts a point from latitude/longitude WGS-84 coordinates (in degrees) into pixel XY
coordinates at a specified level of detail.
/// </summary>
/// <param name="position">Position coordinate in the format [longitude, latitude]</param>
/// <param name="zoom">Zoom level.</param>
/// <param name="tileSize">The size of the tiles in the tile pyramid.</param>
/// <returns>A global pixel coordinate.</returns>
public static double[] PositionToGlobalPixel(double[] position, int zoom, int tileSize)
{
    var latitude = Clip(position[1], MinLatitude, MaxLatitude);
    var longitude = Clip(position[0], MinLongitude, MaxLongitude);

    var x = (longitude + 180) / 360;
    var sinLatitude = Math.Sin(latitude * Math.PI / 180);
    var y = 0.5 - Math.Log((1 + sinLatitude) / (1 - sinLatitude)) / (4 * Math.PI);

    var mapSize = MapSize(zoom, tileSize);

    return new double[] {
        Clip(x * mapSize + 0.5, 0, mapSize - 1),
        Clip(y * mapSize + 0.5, 0, mapSize - 1)
    };
}

/// <summary>
/// Converts pixel XY coordinates into tile XY coordinates of the tile containing the specified
pixel.
/// </summary>
/// <param name="pixel">Pixel coordinates in the format of [x, y].</param>
/// <param name="tileSize">The size of the tiles in the tile pyramid.</param>
/// <param name="tileX">Output parameter receiving the tile X coordinate.</param>
/// <param name="tileY">Output parameter receiving the tile Y coordinate.</param>
public static void GlobalPixelToTileXY(double[] pixel, int tileSize, out int tileX, out int tileY)
{
    tileX = (int)(pixel[0] / tileSize);
    tileY = (int)(pixel[1] / tileSize);
}

/// <summary>
/// Performs a scale transform on a global pixel value from one zoom level to another.
/// </summary>
/// <param name="pixel">Pixel coordinates in the format of [x, y].</param>
/// <param name="oldZoom">The zoom level in which the input global pixel value is from.</param>
/// <returns>A scale pixel coordinate.</returns>
public static double[] ScaleGlobalPixel(double[] pixel, double oldZoom, double newZoom)
{
    var scale = Math.Pow(2, oldZoom - newZoom);

    return new double[] { pixel[0] * scale, pixel[1] * scale };
}

/// <summary>
/// Performs a scale transform on a set of global pixel values from one zoom level to another.
/// </summary>
/// <param name="pixels">A set of global pixel value from the old zoom level. Points are in the

```

```

    ///> [x,y].</param>
    ///> <param name="oldZoom">The zoom level in which the input global pixel values is from.</param>
    ///> <param name="newZoom">The new zoom level in which the output global pixel values should be
aligned with.</param>
    ///> <returns>A set of global pixel values that has been scaled for the new zoom level.</returns>
public static double[][] ScaleGlobalPixels(double[][] pixels, double oldZoom, double newZoom)
{
    var scale = Math.Pow(2, oldZoom - newZoom);

    var output = new System.Collections.Generic.List<double[]>();
    foreach (var p in pixels)
    {
        output.Add(new double[] { p[0] * scale, p[1] * scale });
    }

    return output.ToArray();
}

///> <summary>
///> Converts tile XY coordinates into a global pixel XY coordinates of the upper-left pixel of the
specified tile.
///> </summary>
///> <param name="tileX">Tile X coordinate.</param>
///> <param name="tileY">Tile Y coordinate.</param>
///> <param name="tileSize">The size of the tiles in the tile pyramid.</param>
///> <param name="pixelX">Output parameter receiving the X coordinate of the point, in pixels.
</param>
///> <param name="pixelY">Output parameter receiving the Y coordinate of the point, in pixels.
</param>
public static double[] TileXYToGlobalPixel(int tileX, int tileY, int tileSize)
{
    return new double[] { tileX * tileSize, tileY * tileSize };
}

///> <summary>
///> Converts tile XY coordinates into a quadkey at a specified level of detail.
///> </summary>
///> <param name="tileX">Tile X coordinate.</param>
///> <param name="tileY">Tile Y coordinate.</param>
///> <param name="zoom">Zoom level</param>
///> <returns>A string containing the quadkey.</returns>
public static string TileXYToQuadKey(int tileX, int tileY, int zoom)
{
    var quadKey = new StringBuilder();
    for (int i = zoom; i > 0; i--)
    {
        char digit = '0';
        int mask = 1 << (i - 1);
        if ((tileX & mask) != 0)
        {
            digit++;
        }
        if ((tileY & mask) != 0)
        {
            digit++;
            digit++;
        }
        quadKey.Append(digit);
    }
    return quadKey.ToString();
}

///> <summary>
///> Converts a quadkey into tile XY coordinates.
///> </summary>
///> <param name="quadKey">Quadkey of the tile.</param>
///> <param name="tileX">Output parameter receiving the tile X coordinate.</param>
///> <param name="tileY">Output parameter receiving the tile Y coordinate.</param>
///> <param name="zoom">Output parameter receiving the zoom level.</param>

```

```

/// <param name="zoom">Output parameter receiving the zoom level.</param>
public static void QuadKeyToTileXY(string quadKey, out int tileX, out int tileY, out int zoom)
{
    tileX = tileY = 0;
    zoom = quadKey.Length;
    for (int i = zoom; i > 0; i--)
    {
        int mask = 1 << (i - 1);
        switch (quadKey[zoom - i])
        {
            case '0':
                break;

            case '1':
                tileX |= mask;
                break;

            case '2':
                tileY |= mask;
                break;

            case '3':
                tileX |= mask;
                tileY |= mask;
                break;

            default:
                throw new ArgumentException("Invalid QuadKey digit sequence.");
        }
    }
}

/// <summary>
/// Calculates the XY tile coordinates that a coordinate falls into for a specific zoom level.
/// </summary>
/// <param name="position">Position coordinate in the format [longitude, latitude]</param>
/// <param name="zoom">Zoom level</param>
/// <param name="tileSize">The size of the tiles in the tile pyramid.</param>
/// <param name="tileX">Output parameter receiving the tile X position.</param>
/// <param name="tileY">Output parameter receiving the tile Y position.</param>
public static void PositionToTileXY(double[] position, int zoom, int tileSize, out int tileX, out
int tileY)
{
    var latitude = Clip(position[1], MinLatitude, MaxLatitude);
    var longitude = Clip(position[0], MinLongitude, MaxLongitude);

    var x = (longitude + 180) / 360;
    var sinLatitude = Math.Sin(latitude * Math.PI / 180);
    var y = 0.5 - Math.Log((1 + sinLatitude) / (1 - sinLatitude)) / (4 * Math.PI);

    //tileSize needed in calculations as in rare cases the multiplying/rounding/dividing can make
    the difference of a pixel which can result in a completely different tile.
    var mapSize = MapSize(zoom, tileSize);
    tileX = (int)Math.Floor(Clip(x * mapSize + 0.5, 0, mapSize - 1) / tileSize);
    tileY = (int)Math.Floor(Clip(y * mapSize + 0.5, 0, mapSize - 1) / tileSize);
}

/// <summary>
/// Calculates the tile quadkey strings that are within a specified viewport.
/// </summary>
/// <param name="position">Position coordinate in the format [longitude, latitude]</param>
/// <param name="zoom">Zoom level</param>
/// <param name="width">The width of the map viewport in pixels.</param>
/// <param name="height">The height of the map viewport in pixels.</param>
/// <param name="tileSize">The size of the tiles in the tile pyramid.</param>
/// <returns>A list of quadkey strings that are within the specified viewport.</returns>
public static string[] GetQuadkeysInView(double[] position, int zoom, int width, int height, int
tileSize)
{
}

```

```

var p = PositionToGlobalPixel(position, zoom, tileSize);

var top = p[1] - height * 0.5;
var left = p[0] - width * 0.5;

var bottom = p[1] + height * 0.5;
var right = p[0] + width * 0.5;

var tl = GlobalPixelToPosition(new double[] { left, top }, zoom, tileSize);
var br = GlobalPixelToPosition(new double[] { right, bottom }, zoom, tileSize);

//Boudning box in the format: [west, south, east, north];
var bounds = new double[] { tl[0], br[1], br[0], tl[1] };

return GetQuadkeysInBoundingBox(bounds, zoom, tileSize);
}

/// <summary>
/// Calculates the tile quadkey strings that are within a bounding box at a specific zoom level.
/// </summary>
/// <param name="bounds">A bounding box defined as an array of numbers in the format of [west,
south, east, north].</param>
/// <param name="zoom">Zoom level to calculate tiles for.</param>
/// <param name="tileSize">The size of the tiles in the tile pyramid.</param>
/// <returns>A list of quadkey strings.</returns>
public static string[] GetQuadkeysInBoundingBox(double[] bounds, int zoom, int tileSize)
{
    var keys = new System.Collections.Generic.List<string>();

    if (bounds != null && bounds.Length >= 4)
    {
        PositionToTileXY(new double[] { bounds[3], bounds[0] }, zoom, tileSize, out int tlX, out int
tlY);
        PositionToTileXY(new double[] { bounds[1], bounds[2] }, zoom, tileSize, out int brX, out int
brY);

        for (int x = tlX; x <= brX; x++)
        {
            for (int y = tlY; y <= brY; y++)
            {
                keys.Add(TileXYToQuadKey(x, y, zoom));
            }
        }
    }

    return keys.ToArray();
}

/// <summary>
/// Calculates the bounding box of a tile.
/// </summary>
/// <param name="tileX">Tile X coordinate</param>
/// <param name="tileY">Tile Y coordinate</param>
/// <param name="zoom">Zoom level</param>
/// <param name="tileSize">The size of the tiles in the tile pyramid.</param>
/// <returns>A bounding box of the tile defined as an array of numbers in the format of [west,
south, east, north].</returns>
public static double[] TileXYToBoundingBox(int tileX, int tileY, double zoom, int tileSize)
{
    //Top left corner pixel coordinates
    var x1 = (double)(tileX * tileSize);
    var y1 = (double)(tileY * tileSize);

    //Bottom right corner pixel coordinates
    var x2 = (double)(x1 + tileSize);
    var y2 = (double)(y1 + tileSize);

    var nw = GlobalPixelToPosition(new double[] { x1, y1 }, zoom, tileSize);
    var se = GlobalPixelToPosition(new double[] { x2, y2 }, zoom, tileSize);
}

```

```

        return new double[] { nw[0], se[1], se[0], nw[1] };
    }

    /// <summary>
    /// Calculates the best map view (center, zoom) for a bounding box on a map.
    /// </summary>
    /// <param name="bounds">A bounding box defined as an array of numbers in the format of [west,
    south, east, north].</param>
    /// <param name="mapWidth">Map width in pixels.</param>
    /// <param name="mapHeight">Map height in pixels.</param>
    /// <param name="padding">Width in pixels to use to create a buffer around the map. This is to keep
    markers from being cut off on the edge</param>
    /// <param name="tileSize">The size of the tiles in the tile pyramid.</param>
    /// <param name="latitude">Output parameter receiving the center latitude coordinate.</param>
    /// <param name="longitude">Output parameter receiving the center longitude coordinate.</param>
    /// <param name="zoom">Output parameter receiving the zoom level</param>
    public static void BestMapView(double[] bounds, double mapWidth, double mapHeight, int padding, int
tileSize, out double centerLat, out double centerLon, out double zoom)
{
    if (bounds == null || bounds.Length < 4)
    {
        centerLat = 0;
        centerLon = 0;
        zoom = 1;
        return;
    }

    double boundsDeltaX;

    //Check if east value is greater than west value which would indicate that bounding box crosses
    the antimeridian.
    if (bounds[2] > bounds[0])
    {
        boundsDeltaX = bounds[2] - bounds[0];
        centerLon = (bounds[2] + bounds[0]) / 2;
    }
    else
    {
        boundsDeltaX = 360 - (bounds[0] - bounds[2]);
        centerLon = ((bounds[2] + bounds[0]) / 2 + 360) % 360 - 180;
    }

    var ry1 = Math.Log((Math.Sin(bounds[1] * Math.PI / 180) + 1) / Math.Cos(bounds[1] * Math.PI /
180));
    var ry2 = Math.Log((Math.Sin(bounds[3] * Math.PI / 180) + 1) / Math.Cos(bounds[3] * Math.PI /
180));
    var ryc = (ry1 + ry2) / 2;

    centerLat = Math.Atan(Math.Sinh(ryc)) * 180 / Math.PI;

    var resolutionHorizontal = boundsDeltaX / (mapWidth - padding * 2);

    var vy0 = Math.Log(Math.Tan(Math.PI * (0.25 + centerLat / 360)));
    var vy1 = Math.Log(Math.Tan(Math.PI * (0.25 + bounds[3] / 360)));
    var zoomFactorPowered = (mapHeight * 0.5 - padding) / (40.7436654315252 * (vy1 - vy0));
    var resolutionVertical = 360.0 / (zoomFactorPowered * tileSize);

    var resolution = Math.Max(resolutionHorizontal, resolutionVertical);

    zoom = Math.Log(360 / (resolution * tileSize), 2);
}
}
}

```

NOTE

The interactive map controls in the Azure Maps SDK's have helper functions for converting between geospatial positions and viewport pixels.

- [Web SDK: Map pixel and position calculations](#)

Next steps

Directly access map tiles from the Azure Maps REST services:

[Get map tiles](#)

[Get traffic flow tiles](#)

[Get traffic incident tiles](#)

Learn more about geospatial concepts:

[Azure Maps glossary](#)

Data structures in Azure Maps Mobility services (Preview)

3/5/2021 • 2 minutes to read • [Edit Online](#)

IMPORTANT

Azure Maps Mobility services are currently in public preview. This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

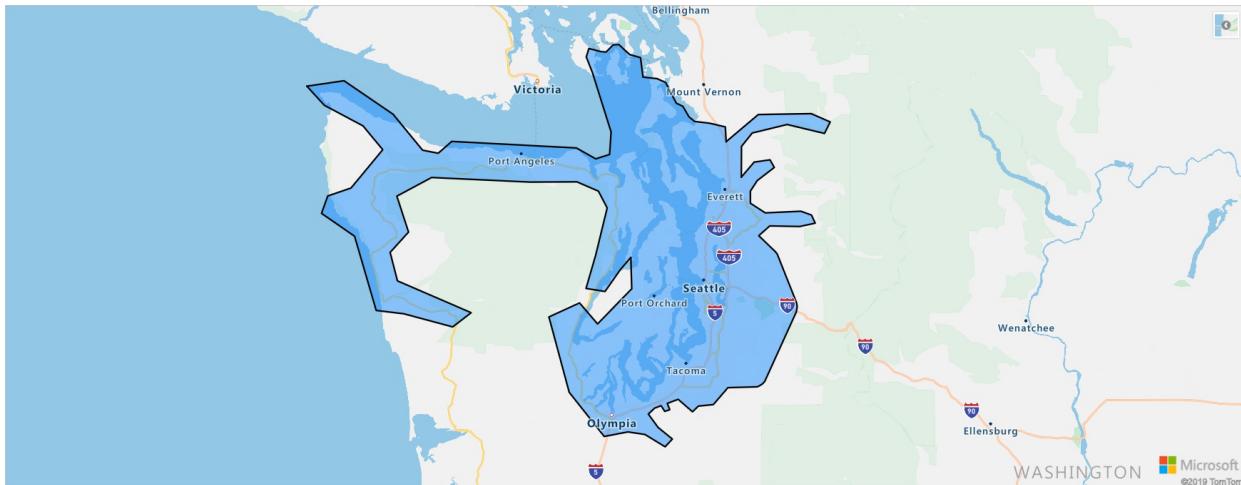
This article introduces the concept of Metro Area in [Azure Maps Mobility services](#). We discuss some of common fields that are returned when this service is queried for public transit stops and lines. We recommend reading this article before developing with the Mobility services APIs.

Metro area

Mobility services (Preview) data is grouped by supported metro areas. Metro areas don't follow city boundaries. A metro area can contain multiple cities, densely populated city, and surrounding cities. In fact, a country/region can be one metro area.

The `metroID` is a metro area's ID that can be used to call the [Get Metro Area Info API](#). Use Azure Maps' "Get Metro" API to request transit types, transit agencies, active alerts, and additional details for the chosen metro. You can also request the supported metro areas and metroIDs. Metro area IDs are subject to change.

metroID: 522 **Name:** Seattle-Tacoma-Bellevue



Stop IDs

Transit stops can be referred to by two types of IDs, the [General Transit Feed Specification \(GTFS\)](#) ID and the Azure Maps stop ID. The GTFS ID is referred to as the `stopKey` and the Azure Maps stop ID is referred to as `stopID`. When frequently referring to transit stops, you're encouraged to use the Azure Maps stop ID. `stopID` is more stable and likely to stay the same as long as the physical stop exists. The GTFS stop ID is updated more often. For example, GTFS stop ID can be updated per the GTFS provider request or when a new GTFS version is released. Although the physical stop had no change, the GTFS stop ID may change.

To start, you can request nearby transit stops using [Get Nearby Transit API](#).

Line Groups and Lines

Mobility services (Preview) use a parallel data model for Lines and Line Groups. This model is used to better deal with changes inherited from [GTFS](#) routes and the trips data.

Line Groups

A Line Group is an entity, which groups together all lines that are logically part of the same group. Usually, a line group contains two lines, one from point A to B, and the other returning from point B to A. Both lines would belong to the same Public Transport agency and have the same line number. However, there may be cases in which a line group has more than two lines or only a single line within it.

Lines

As discussed above, each line group is composed of a set of lines. Each line group is composed of two lines, and each line describes a direction. However, there are cases in which more lines compose a line group. For example, there's a line that sometimes detours through a certain neighborhood and sometimes doesn't. In both cases, it operates under the same line number. Also a line group can be composed of a single line. A circular line with a single direction is a line group with one line.

To begin, you can request line groups by using the [Get Transit Line API](#).

Next steps

Learn how to request transit data using Mobility services (Preview):

[How to request transit data](#)

Learn how to request real-time data using Mobility services (Preview):

[How to request real-time data](#)

Explore the Azure Maps Mobility services (Preview) API documentation

[Mobility services API documentation](#)

Weather services in Azure Maps

6/8/2021 • 11 minutes to read • [Edit Online](#)

This article introduces concepts that apply to Azure Maps [Weather services](#). We recommend going through this article before starting out with the weather APIs.

Unit types

Some of the Weather service (Preview) APIs allow user to specify if the data is returned either in metric or in imperial units. The returned responses for these APIs include unitType and a numeric value that can be used for unit translations. See table below to interpret these values.

UNITTYPE	DESCRIPTION
0	feet
1	inches
2	miles
3	millimeter
4	centimeter
5	meter
6	kilometer
7	kilometersPerHour
8	knots
9	milesPerHour
10	metersPerSecond
11	hectoPascals
12	inchesOfMercury
13	kiloPascals
14	millibars
15	millimetersOfMercury
16	poundsPerSquareInch
17	celsius

UNIT	TYPE	DESCRIPTION
18		fahrenheit
19		kelvin
20		percent
21		float
22		integer
31		MicrogramsPerCubicMeterOfAir

Weather icons

Some of the Weather service (Preview) APIs return the `iconCode` in the response. The `iconCode` is a numeric value used to define the icon. Don't link directly to these images from your applications, the URLs can and will change.

ICON NUMBER	ICON	DAY	NIGHT	TEXT
1		Yes	No	Sunny
2		Yes	No	Mostly Sunny
3		Yes	No	Partly Sunny
4		Yes	No	Intermittent Clouds
5		Yes	No	Hazy Sunshine
6		Yes	No	Mostly Cloudy
7		Yes	Yes	Cloudy
8		Yes	Yes	Dreary (Overcast)
11		Yes	Yes	Fog

ICON NUMBER	ICON	DAY	NIGHT	TEXT
12		Yes	Yes	Showers
13		Yes	No	Mostly Cloudy with Showers
14		Yes	No	Partly Sunny with Showers
15		Yes	Yes	Thunderstorms
16		Yes	No	Mostly Cloudy with Thunderstorms
17		Yes	No	Partly Sunny with Thunderstorms
18		Yes	Yes	Rain
19		Yes	Yes	Flurries
20		Yes	No	Mostly Cloudy with Flurries
21		Yes	No	Partly Sunny with Flurries
22		Yes	Yes	Snow
23		Yes	No	Mostly Cloudy with Snow
24		Yes	Yes	Ice
25		Yes	Yes	Sleet
26		Yes	Yes	Freezing Rain

ICON NUMBER	ICON	DAY	NIGHT	TEXT
29	 A black outline icon showing a water droplet above a snowflake.	Yes	Yes	Rain and Snow
30	 A black outline icon of a thermometer with a single scale line near the top.	Yes	Yes	Hot
31	 A black outline icon of a thermometer with a single scale line near the bottom.	Yes	Yes	Cold
32	 A black outline icon showing two curved arrows pointing in opposite directions, indicating wind.	Yes	Yes	Windy
33	 A black outline icon of a crescent moon.	No	Yes	Clear
34	 A black outline icon of a crescent moon with a small cloud symbol to its right.	No	Yes	Mostly Clear
35	 A black outline icon of a crescent moon with a larger cloud symbol to its right.	No	Yes	Partly Cloudy
36	 A black outline icon of a crescent moon with several small cloud symbols around it.	No	Yes	Intermittent Clouds
37	 A black outline icon of a crescent moon with wavy lines below it, suggesting haze or light reflection.	No	Yes	Hazy Moonlight
38	 A black outline icon of a crescent moon with a large, prominent cloud symbol to its right.	No	Yes	Mostly Cloudy
39	 A black outline icon of a crescent moon with a small cloud symbol and a single raindrop falling from it.	No	Yes	Partly Cloudy with Showers
40	 A black outline icon of a crescent moon with a large, dark cloud symbol and multiple raindrops falling from it.	No	Yes	Mostly Cloudy with Showers
41	 A black outline icon of a crescent moon with a large, dark cloud symbol and a lightning bolt striking from it.	No	Yes	Partly Cloudy with Thunderstorms
42	 A black outline icon of a crescent moon with a large, dark cloud symbol containing a lightning bolt.	No	Yes	Mostly Cloudy with Thunderstorms
43	 A black outline icon of a crescent moon with a large, dark cloud symbol containing a snowflake.	No	Yes	Mostly Cloudy with Flurries

ICON NUMBER	ICON	DAY	NIGHT	TEXT
44		No	Yes	Mostly Cloudy with Snow

Radar and satellite imagery color scale

Via [Get Map Tile v2 API](#) users can request latest radar and infrared satellite images. See below guide to help interpret colors used for radar and satellite tiles.

Radar Images

The table below provides guidance to interpret the radar images and create a map legend for Radar tile data.

HEX COLOR CODE	COLOR SAMPLE	WEATHER CONDITION
#93c701		Rain-Light
#ffd701		Rain-Moderate
#f05514		Rain-Heavy
#dc250e		Rain-Severe
#9ec8f2		Snow-Light
#2a8fdb		Snow-Moderate
#144bed		Snow-Heavy
#020096		Snow-Severe
#e6a5c8		Ice-Light
#d24fa0		Ice-Moderate
#b71691		Ice-Severe
#7a1570		Ice-Heavy
#c196e6		Mix-Light
#ae6ee6		Mix-Moderate
#8a32d7		Mix-Heavy
#6500ba		Mix-Severe

Detailed color palette for radar tiles with Hex color codes and dBZ values is shown below. dBZ represents precipitation intensity in weather radar.

RAIN	ICE	SNOW	MIXED
dBZ (color)	dBZ (color)	dBZ (color)	dBZ (color)
1.25 (#93C701)	1.25 (#E6A5C8)	1.25 (#9EC8F2)	1.25 (#C196E6)
2.5 (#92C201)	2.5 (#E6A2C6)	2.5 (#98C5F0)	2.5 (#BF92E6)
3.75 (#92BE01)	3.75 (#E69FC5)	3.75 (#93C3EF)	3.75 (#BD8EE6)
5 (#92BA02)	5 (#E69DC4)	5 (#8DC1EE)	5 (#BB8BE6)
6.25 (#92B502)	6.25 (#E69AC2)	6.25 (#88BFEC)	6.25 (#BA87E6)
6.75 (#92B403)	7.5 (#E697C1)	7.5 (#82BDEB)	7.5 (#B883E6)
8 (#80AD02)	8.75 (#E695C0)	8.75 (#7DBAEA)	8.75 (#B680E6)
9.25 (#6FA602)	10 (#E692BE)	10 (#77B8E8)	10 (#B47CE6)
10.5 (#5EA002)	11.25 (#E68FBD)	11.25 (#72B6E7)	11.25 (#B378E6)
11.75 (#4D9902)	12.5 (#E68DBC)	12.5 (#6CB4E6)	12.5 (#B175E6)
12.25 (#479702)	13.75 (#E68ABA)	13.75 (#67B2E5)	13.75 (#AF71E6)
13.5 (#3D9202)	15 (#E687B9)	15 (#61AEE4)	15 (#AE6EE6)
14.75 (#338D02)	16.25 (#E685B8)	16.25 (#5BABE3)	16.25 (#AB6AE4)
16 (#298802)	17.5 (#E682B6)	17.5 (#56A8E2)	17.5 (#A967E3)
17.25 (#1F8302)	18.75 (#E67FB5)	18.75 (#50A5E1)	18.75 (#A764E2)
17.75 (#1B8103)	20 (#E67DB4)	20 (#4BA2E0)	20 (#A560E1)
19 (#187102)	21.25 (#E275B0)	21.25 (#459EDF)	21.25 (#A35DE0)
20.25 (#166102)	22.5 (#DF6DAD)	22.5 (#409BDE)	22.5 (#A15ADF)
20.75 (#165B02)	23.75 (#DC66AA)	23.75 (#3A98DD)	23.75 (#9F56DE)
22 (#135001)	25 (#D85EA6)	25 (#3595DC)	25 (#9D53DD)
23.25 (#114501)	26.25 (#D556A3)	26.25 (#2F92DB)	26.25 (#9B50DC)
24.5 (#0F3A01)	27.5 (#D24FA0)	27.5 (#2A8FDB)	27.5 (#9648DA)
25.75 (#124C01)	28.75 (#CE479E)	28.75 (#2581DE)	28.75 (#9241D9)
27 (#114401)	30 (#CB409C)	30 (#2173E2)	30 (#8E39D8)

RAIN	ICE	SNOW	MIXED
28.25 (#0F3D01)	31.25 (#C7399A)	31.25 (#1C66E5)	31.25 (#8A32D7)
28.75 (#0F3A01)	32.5 (#C43298)	32.5 (#1858E9)	32.5 (#862ED2)
30 (#375401)	33.75 (#C12B96)	33.75 (#144BED)	33.75 (#832BCE)
31.25 (#5F6E01)	35 (#BD2494)	35 (#1348EA)	35 (#7F28C9)
32.5 (#878801)	36.25 (#BA1D92)	36.25 (#1246E7)	36.25 (#7C25C5)
33.75 (#AFA201)	37.5 (#B71691)	37.5 (#1144E4)	37.5 (#7822C1)
35 (#D7BC01)	38.75 (#B51690)	38.75 (#1142E1)	38.75 (#751FBC)
36.25 (#FFD701)	40 (#B3168F)	40 (#1040DE)	40 (#711CB8)
37.5 (#FEB805)	41.25 (#B1168E)	41.25 (#0F3EDB)	41.25 (#6E19B4)
38.75 (#FCAB06)	42.5 (#AF168D)	42.5 (#0F3CD8)	42.5 (#6D18B4)
40 (#FA9E07)	43.75 (#AD168C)	43.75 (#0E3AD5)	43.75 (#6D17B4)
41.25 (#F89209)	45 (#AB168B)	45 (#0D38D2)	45 (#6D16B4)
42.5 (#F05514)	46.25 (#A9168A)	46.25 (#0C36CF)	46.25 (#6C15B4)
43.75 (#E74111)	47.5 (#A81689)	47.5 (#0C34CC)	47.5 (#6C14B5)
45 (#DF2D0F)	48.75 (#A61688)	48.75 (#0B32C9)	48.75 (#6C13B5)
45.5 (#DC250E)	50 (#A41687)	50 (#0A30C6)	50 (#6B12B5)
46.75 (#D21C0C)	51.25 (#A21686)	51.25 (#0A2EC4)	51.25 (#6B11B5)
48 (#C9140A)	52.5 (#A01685)	52.5 (#092BC1)	52.5 (#6B10B6)
49.25 (#BF0C09)	53.75 (#9E1684)	53.75 (#0929BF)	53.75 (#6A0FB6)
50 (#BA0808)	55 (#9C1683)	55 (#0826BC)	55 (#6A0EB6)
56.25 (#6f031b)	56.25 (#9B1682)	56.25 (#0824BA)	56.25 (#6A0DB6)
57.5 (#9f0143)	57.5 (#981580)	57.5 (#0721B7)	57.5 (#690CB6)
58.75 (#c10060)	58.75 (#96157F)	58.75 (#071FB5)	58.75 (#690CB7)
60 (#e70086)	60 (#94157E)	60 (#071DB3)	60 (#690BB7)
61.25 (#e205a0)	61.25 (#92157D)	61.25 (#061AB0)	61.25 (#680AB7)

RAIN	ICE	SNOW	MIXED
62.5 (#cc09ac)	62.5 (#90157C)	62.5 (#0618AE)	62.5 (#6809B7)
63.75 (#b50eb7)	63.75 (#8D157A)	63.75 (#0515AB)	63.75 (#6808B8)
65 (#9315c8)	65 (#8B1579)	65 (#0513A9)	65 (#6707B8)
66.25 (#8f21cc)	66.25 (#891578)	66.25 (#0410A6)	66.25 (#6706B8)
67.5 (#983acb)	67.5 (#871577)	67.5 (#040EA4)	67.5 (#6705B8)
68.75 (#9d49cb)	68.75 (#851576)	68.75 (#040CA2)	68.75 (#6604B8)
70 (#a661ca)	70 (#821574)	70 (#03099F)	70 (#6603B9)
71.25 (#ad72c9)	71.25 (#801573)	71.25 (#03079D)	71.25 (#6602B9)
72.5 (#b78bc6)	72.5 (#7E1572)	72.5 (#02049A)	72.5 (#6501B9)
73.75 (#bf9bc4)	73.75 (#7C1571)	73.75 (#020298)	73.75 (#6500B9)
75 (#c9b5c2)	75 (#7A1570)	75 (#020096)	75 (#6500BA)

Satellite Images

The table below provides guidance to interpret the infrared satellite images showing clouds by their temperature and how to create a map legend for these tiles.

HEX COLOR CODE	COLOR SAMPLE	CLOUD TEMPERATURE
#b5b5b5		Temperature-Low
#d24fa0		
#8a32d7		
#144bed		
#479702		
#72b403		
#93c701		
#ffd701		
#f05514		
#dc250e		
#ba0808		

HEX COLOR CODE	COLOR SAMPLE	CLOUD TEMPERATURE
#1f1f1f		Temperature-High

Detailed color palette for infrared satellite tiles is shown below.

TEMP (K)	HEX COLOR CODE
198	#fe050505
198.43	#fe120505
198.87	#fc1f0505
199.3	#fc2c0606
199.74	#fa390606
200.17	#fa460606
200.61	#f8530606
201.04	#f8600707
201.48	#f66c0707
201.91	#f6790707
202.35	#f4860707
202.78	#f4930707
203.22	#f2a00808
203.65	#f2ad0808
204.09	#f0ba0808
204.52	#f0bd0a09
204.96	#eec00d09
205.39	#eec30f0a
205.83	#ecc5120a
206.26	#ecc8140b
206.7	#eacb170b
207.13	#eace190c

TEMP (K)	HEX COLOR CODE
207.57	#e8d11b0c
208	#e8d41e0d
208.43	#e6d6200d
208.87	#e6d9230e
209.3	#e4dc250e
209.74	#e4df2c0f
210.17	#e2e23310
210.61	#e2e53a11
211.04	#e0e73a11
211.48	#e0ea4712
211.91	#deed4e13
212.35	#def05514
212.78	#dcf15e13
213.22	#dcf26811
213.65	#daf37110
214.09	#daf47a0f
214.52	#d8f5830d
214.96	#d8f68d0c
215.39	#d6f8960b
215.83	#d6f99f09
216.26	#d4faa908
216.7	#d4fbdb206
217.13	#d2fcbb05
217.57	#d2fdc404
218	#d0fece02

TEMP (K)	HEX COLOR CODE
218.43	#d0ffd701
218.87	#cef8d601
219.3	#cef2d501
219.74	#ccebd401
220.17	#cce4d301
220.61	#caddd201
221.04	#cad7d101
221.48	#c8d0d001
221.91	#c8c9cf01
222.35	#c6c2ce01
222.78	#c6bccd01
223.22	#c4b5cc01
223.65	#c4aecb01
224.09	#c2a7ca01
224.52	#c2a1c901
224.96	#c09ac801
225.39	#c093c701
225.83	#be90c501
226.26	#be8ec401
226.7	#bc8bc202
227.13	#bc88c102
227.57	#ba85bf02
228	#ba83be02
228.43	#b880bc02
228.87	#b87dba02

TEMP (K)	HEX COLOR CODE
229.3	#b678b703
229.74	#b675b603
230.17	#b472b403
230.61	#b46eb203
231.04	#b26bb203
231.48	#b267ad03
231.97	#b064aa03
232.35	#b060a803
232.78	#ae5da603
233.22	#ae59a302
233.65	#ac55a102
234.09	#ac529e02
234.52	#aa4e9c02
234.96	#aa4b9902
235.39	#a8479702
235.83	#a845940b
236.26	#a6439115
236.7	#a6418e1e
237.13	#a43f8b28
237.57	#a43d8831
238	#a23b853a
238.43	#a2398244
238.87	#a0377f4d
239.3	#a0357c57
239.74	#9e337960

TEMP (K)	HEX COLOR CODE
240.17	#9e317669
240.61	#9c2f7373
241.04	#9c2c6f7c
241.48	#9a2a6c86
241.91	#9a28698f
242.35	#98266698
242.78	#982463a2
243.22	#962260ab
243.65	#96205db5
244.09	#941e5abe
244.52	#941c57c7
244.85	#921a54d1
245.39	#921851da
245.83	#90164ee4
246.26	#90144bed
246.7	#8e2148eb
247.13	#8e2e45e8
247.57	#8c3b43e6
248	#8c4840e3
248.43	#8a563de1
248.87	#8a633ade
249.3	#887038dc
249.74	#887d35d9
250.17	#868a32d7
250.61	#869034d2

TEMP (K)	HEX COLOR CODE
251.04	#849637ce
251.48	#849c39c9
251.91	#82a23cc5
252.35	#82a83ec0
252.78	#80ae41bc
253.22	#80b443b7
253.65	#7eba45b2
253.09	#7ec048ae
254.52	#7cc64aa9
254.96	#7ccc4da5
255.39	#7ad24fa0
255.83	#7ad85fac
256.26	#78dd6eb8
256.7	#78e37ec4
257.13	#76e98ed0
257.57	#76ee9ddb
258	#74f4ade7
258.43	#74f9bcf3
258.87	#72ffccff
259.3	#71ffffff
259.74	#71fcfcfc
260.17	#6ff6f6f6
260.61	#6ff6f6f6
261.04	#6df3f3f3
261.48	#6df3f3f3

TEMP (K)	HEX COLOR CODE
261.91	#6beded
262.35	#6beded
262.78	#69e7e7e7
263.22	#69e7e7e7
263.65	#67e1e1e1
264.09	#67e1e1e1
264.52	#65dedede
264.96	#65dedede
265.39	#63d8d8d8
265.83	#63d8d8d8
265.84	#61d1d1d1
266.26	#61d1d1d1
267.13	#5fcecece
267.57	#5fcecece
268	#5dc8c8c8
268.43	#5dc8c8c8
268.87	#5bc2c2c2
269.3	#5bc2c2c2
269.74	#59bcbcbc
270.17	#59bcbcbc
270.61	#57b9b9b9
271.04	#57b9b9b9
271.48	#55b3b3b3
271.91	#55b3b3b3
272.35	#53adadad

TEMP (K)	HEX COLOR CODE
272.78	#53adadad
273.22	#51aaaaaa
273.65	#51aaaaaa
274.09	#4fa4a4a4
274.52	#4fa4a4a4
274.96	#4d9e9e9e
275.39	#4d9e9e9e
275.83	#4b989898
276.26	#4b989898
276.7	#49959595
277.13	#49959595
277.57	#478f8f8f
278	#478f8f8f
278.43	#45898989
278.87	#45898989
279.2	#43868686
279.74	#43868686
280.17	#417f7f7f
280.61	#417f7f7f
281.04	#3f797979
281.48	#3f797979
281.91	#3d737373
282.35	#3d737373
282.78	#3b707070
283.22	#3b707070

TEMP (K)	HEX COLOR CODE
283.65	#396a6a6a
284.09	#396a6a6a
284.52	#37646464
284.96	#37646464
285.39	#35616161
285.83	#35616161
286.26	#335b5b5b
286.7	#335b5b5b
287.13	#31555555
287.57	#31555555
288	#2f4f4f4f
288.43	#2f4f4f4f
288.87	#2d4c4c4c
289.3	#2d4c4c4c
289.74	#2b464646
290.17	#2b464646
290.61	#29404040
291.04	#29404040
291.48	#273d3d3d
291.91	#273d3d3d
292.35	#25373737
292.78	#25373737
293.22	#23313131
293.65	#23313131
294.09	#212a2a2a

TEMP (K)	HEX COLOR CODE
294.52	#212a2a2a
294.96	#1f272727
295.39	#1f272727
295.83	#1d212121
296.26	#1d212121
296.7	#1b1b1b1b
297.13	#1b1b1b1b
297.57	#19181818
298	#19181818
298.43	#17121212
298.87	#17121212
299.3	#150c0c0c
299.74	#150c0c0c
300.17	#13060606
300.61	#13060606
301.04	#11000000
301.48	#11000000
301.91	#0f797979
302.35	#0f797979
302.78	#0d737373
303.22	#0d737373
303.65	#0b6d6d6d
304.09	#0b6d6d6d
304.52	#09676767
304.92	#09676767

TEMP (K)	HEX COLOR CODE
305.39	#07616161
305.83	#07616161
306.26	#055b5b5b
306.7	#055b5b5b
307.13	#02555555
307.57	#02555555
308	#00525252
308	#00525252

Index IDs and Index Groups IDs

[Get Daily Indices API](#) allows users to restrict returned results to specific index types or index groups.

Below is a table of available index IDs, their names, and a link to their range sets. Below this table is a table listing the various index groups.

INDEX NAME	ID	VALUE RANGE
Arthritis Pain	21	Beneficial-At Extreme Risk
Asthma	23	Beneficial-At Extreme Risk
Beach & Pool	10	Poor-Excellent 1
Bicycling	4	Poor-Excellent 1
Common Cold	25	Beneficial-At Extreme Risk
Composting	38	Poor-Excellent 1
Construction	14	Poor-Excellent 1
COPD	44	Beneficial-At Extreme Risk
Dog Walking Comfort	43	Poor-Excellent 1
Driving	40	Poor-Excellent 2
Dust & Dander	18	Low-Extreme 1
Field Readiness	32	Poor-Excellent 1
Fishing	13	Poor-Excellent 1

INDEX NAME	ID	VALUE RANGE
Flight Delays	-3	Very Unlikely-Very Likely 2
Flu	26	Beneficial-At Extreme Risk
Flying Travel Index	31	Excellent-Poor
Fuel Economy	37	Poor-Excellent 1
Golf Weather	5	Poor-Excellent 1
Grass Growing	33	Poor-Excellent 1
Hair Frizz	42	Unlikely-Emergency
Healthy Heart Fitness	16	Poor-Excellent 1
Hiking	3	Poor-Excellent 1
Home Energy Efficiency	36	Poor-Excellent 1
Hunting	20	Poor-Excellent 1
Indoor Activity	-2	Poor-Excellent 1
Jogging	2	Poor-Excellent 1
Kite Flying	9	Poor-Excellent 1
Lawn Mowing	28	Poor-Excellent 1
Migraine Headache	27	Beneficial-At Extreme Risk
Morning School Bus	35	Poor-Excellent 1
Mosquito Activity	17	Low-Extreme 1
Outdoor Activity	29	Poor-Excellent 1
Outdoor Barbecue	24	Poor-Excellent 1
Outdoor Concert	8	Poor-Excellent 1
Running	1	Poor-Excellent 1
Tennis	6	Poor-Excellent 1
Thirst	41	Low-Extreme 2
Sailing	11	Poor-Excellent 1

INDEX NAME	ID	VALUE RANGE
Shopping	39	Poor-Excellent 1
Sinus Headache	30	Beneficial-At Extreme Risk
Skateboarding	7	Poor-Excellent 1
Ski Weather	15	Poor-Excellent 1
Snow Days	19	Very Unlikely-Very Likely
Soil Moisture	34	Poor-Excellent 1
Stargazing	12	Poor-Excellent 1

Below is the list of available Index groups (indexGroupId):

ID	GROUP NAME	INDICES IN THIS GROUP
1	All	All
2	Aches and Pains	Arthritis Pain (21) Migraine Headache (27) Sinus Headache (30)
3	Respiratory	Asthma (23) Common Cold (25) Flu Forecast (26)
4	Gardening	Field Readiness (32) Lawn Mowing (28) Soil Moisture (34)
5	Environmental	Composting (38) Home Energy Efficiency (36) Fuel Economy (37)
6	Outdoor Living	Outdoor Barbecue (24) Mosquito Activity (17)
7	Beach and Marine	Beach & Pool (10) Fishing (13) Sailing (11)
8	Sportsman	Fishing (13) Hunting (20) Outdoor Activity (29)
9	Farming	Field Readiness (32) Soil Moisture (34)

ID	GROUP NAME	INDICES IN THIS GROUP
10	Health	Arthritis Pain (21) Asthma (23) Common Cold (25) Dust & Dander (18) Flu (26) Healthy Heart Fitness (16) Migraine Headache (27)
11	Outdoor	Outdoor Barbecue (24) Beach & Pool (10) Bicycling (4) Outdoor Concert (8) Field Readiness (32) Fishing (13) Golf Weather (5) Hiking (3) Hunting (20) Jogging (2) Kite Flying (9) Mosquito Activity (17) Lawn Mowing (28) Outdoor Activity (29) Running (1) Sailing (11) Skateboarding (7) Ski Weather (15) Soil Moisture (34) Stargazing (12) Tennis (6)
12	Sporting	Bicycling (4) Golf Weather (5) Hiking (3) Jogging (2) Running (1) Skateboarding (7) Ski Weather (15) Tennis (6)
13	Home	Home Energy Efficiency (36) Fuel Economy (37) Indoor Activity (-2)

Daily index range sets

[Get Daily Indices API](#) returns the ranged value and its associated category name for each index ID. Range sets are not the same for all indices. The tables below show the various range sets used by the supported indices listed in [Index IDs and index groups IDs](#). To find out which indices use which range sets, go to the [Index IDs and Index Groups IDs](#) section of this document.

Poor-Excellent 1

CATEGORY NAME	BEGIN RANGE	END RANGE
Poor	0	2.99
Fair	3	4.99

CATEGORY NAME	BEGIN RANGE	END RANGE
Good	5	6.99
Very Good	7	8.99
Excellent	9	10

Poor-Excellent 2

CATEGORY NAME	BEGIN RANGE	END RANGE
Poor	0	3
Fair	3.01	6
Good	6.01	7.5
Very Good	7.51	8.99
Excellent	9	10

Excellent-Poor

CATEGORY NAME	BEGIN RANGE	END RANGE
Excellent	0.00	1.00
Very Good	1.01	3.00
Good	3.01	5.00
Fair	5.01	7.00
Poor	7.01	10.00

Low-Extreme 1

CATEGORY NAME	BEGIN RANGE	END RANGE
Low	0	1.99
Moderate	2	3.99
High	4	5.99
Very High	6	7.99
Extreme	8	10

Low-Extreme 2

CATEGORY NAME	BEGIN RANGE	END RANGE
Low	0	2.99
Moderate	3	4.99
High	5	6.99
Very High	7	8.99
Extreme	9	10

Very Unlikely-Very Likely

CATEGORY NAME	BEGIN RANGE	END RANGE
Very Unlikely	0	1.99
Unlikely	2	3.99
Possibly	4	5.99
Likely	6	7.99
Very Likely	8	10

Very Unlikely-Very Likely 2

CATEGORY NAME	BEGIN RANGE	END RANGE
Very Unlikely	0.00	1.00
Unlikely	1.01	3.00
Possibly	3.01	5.00
Likely	5.01	7.00
Very Likely	7.01	10.00

Unlikely-Emergency

CATEGORY NAME	BEGIN RANGE	END RANGE
Unlikely	0	2.99
Watch	3	4.99
Advisory	5	6.99
Warning	7	8.99
Emergency	9	10

Beneficial-At Extreme Risk

CATEGORY NAME	BEGIN RANGE	END RANGE
Beneficial	0	1.99
Neutral	2	3.99
At Risk	4	5.99
At High Risk	6	7.99
At Extreme Risk	8	10

Azure Maps Weather services frequently asked questions (FAQ)

4/9/2021 • 6 minutes to read • [Edit Online](#)

This article answers to common questions about Azure Maps [Weather services](#) data and features. The following topics are covered:

- Data sources and data models
- Weather services coverage and availability
- Data update frequency
- Developing with Azure Maps SDKs
- Options to visualize weather data, including Microsoft Power BI integration

Data sources and data models

How does Azure Maps source Weather data?

Azure Maps is built with the collaboration of world-class mobility and location technology partners, including AccuWeather, who provides the underlying weather data. To read the announcement of Azure Maps' collaboration with AccuWeather, see [Rain or shine: Azure Maps Weather Services will bring insights to your enterprise.](#)

AccuWeather has real-time weather and environmental information available anywhere in the world, largely due to their partnerships with numerous national governmental weather agencies and other proprietary arrangements. A list of this foundational information is provided below.

- Publicly available global surface observations from government agencies
- Proprietary surface observation datasets from governments and private companies
- High-resolution radar data for over 40 countries/regions
- Best-in-class real-time global lightning data
- Government-issued weather warnings for over 60 countries/regions and territories
- Satellite data from geostationary weather satellites covering the entire world
- Over 150 numerical forecast models including internal, proprietary modeling, government models such as the U.S. Global Forecast System (GFS), and unique downscaled models provided by private companies
- Air quality observations
- Observations from departments of transportation

Tens of thousands of surface observations, along with other data, are incorporated to create and influence the current conditions made available to users. This includes not only freely available standard datasets, but also unique observations obtained from national meteorological services in many countries/regions including India, Brazil, and Canada and other proprietary inputs. These unique datasets increase the spatial and temporal resolution of current condition data for our users.

These datasets are reviewed in real time for accuracy for the Digital Forecast System, which utilizes AccuWeather's proprietary artificial intelligence algorithms to continuously modify the forecasts, ensuring they always incorporate the latest data and thereby maximizing their continual accuracy.

What models create weather forecast data?

Numerous weather forecast guidance systems are utilized to formulate global forecasts. Over 150 numerical

forecast models are used each day, both external and internal datasets. This includes government models such as the European Centre ECMWF and the U.S. Global Forecast System (GFS). Additionally, AccuWeather incorporates proprietary high-resolution models that downscale forecasts to specific locations and strategic regional domains to predict weather with further accuracy. AccuWeather's unique blending and weighting algorithms have been developed over the last several decades. These algorithms optimally leverage the numerous forecast inputs to provide highly accurate forecasts.

Weather services coverage and availability

What kind of coverage can I expect for different countries/regions?

Weather service coverage varies by country/region. All features are not available in every country/region. For more information, see [coverage documentation](#).

Data update frequency

How often is Current Conditions data updated?

Current Conditions data is approximately updated at least once an hour, but can be updated more frequently with rapidly changing conditions – such as large temperature changes, sky conditions changes, precipitation changes, and so on. Most observation stations around the world report many times per hour as conditions change. However, a few areas will still only update once, twice, or four times an hour at scheduled intervals.

Azure Maps caches the Current Conditions data for up to 10 minutes to help capture the near real-time update frequency of the data as it occurs. To see when the cached response expires and avoid displaying outdated data, you can leverage the Expires Header information in the HTTP header of the Azure Maps API response.

How often is Daily and Hourly Forecast data updated?

Daily and Hourly Forecast data is updated multiple times per day, as updated observations are received. For example, if a forecasted high/low temperature is surpassed, our Forecast data will adjust at the next update cycle. This can happen at different intervals but typically happens within an hour. Many sudden weather conditions can cause a forecast data change. For example, on a hot summer afternoon, an isolated thunderstorm can suddenly emerge, bringing heavy cloud coverage and rain. The isolated storm can effectively drop temperature by as much as 10 degrees. This new temperature value will impact the Hourly and Daily Forecasts for the remainder of the day, and as such, will be updated in our datasets.

Azure Maps Forecast APIs are cached for up to 30 mins. To see when the cached response expires and avoid displaying outdated data, you can leverage the Expires Header information in the HTTP header of the Azure Maps API response. We recommend updating as necessary based on a specific product use case and UI (user interface).

Developing with Azure Maps SDKs

Does Azure Maps Web SDK natively support Weather services integration?

The Azure Maps Web SDK provides a services module. The services module is a helper library that makes it easy to use the Azure Maps REST services in web or Node.js applications. by using JavaScript or TypeScript. To get started, see our [documentation](#).

Does Azure Maps Android SDK natively support Weather services integration?

The Azure Maps Android SDKs supports Mercator tile layers, which can have x/y/zoom notation, quad key notation, or EPSG 3857 bounding box notation.

We plan to create a services module for Java/Android similar to the web SDK module. The Android services module will make it easy to access all Azure Maps services in a Java or Android app.

Data visualizations

Does Azure Maps Power BI Visual support Azure Maps weather tiles?

Yes. To learn how to migrate radar and infrared satellite tiles to the Microsoft Power BI visual, see [Add a tile layer to Power BI visual](#).

How do I interpret colors used for radar and satellite tiles?

The Azure Maps [Weather concept article](#) includes a guide to help interpret colors used for radar and satellite tiles. The article covers color samples and HEX color codes.

Can I create radar and satellite tile animations?

Yes. In addition to real-time radar and satellite tiles, Azure Maps customers can request past and future tiles to enhance data visualizations with map overlays. This can be done by directly calling [Get Map Tile v2 API](#) or by requesting tiles via Azure Maps web SDK. Radar tiles are provided for up to 1.5 hours in the past, and for up to 2 hours in the future. The tiles are available in 5-minute intervals. Infrared tiles are provided for up to 3 hours in the past, and are available in 10-minute intervals. For more information, see the open-source Weather Tile Animation [code sample](#).

Do you offer icons for different weather conditions?

Yes. You can find icons and their respective codes [here](#). Notice that only some of the Weather service (Preview) APIs, such as [Get Current Conditions API](#), return the *iconCode* in the response. For more information, see the Current WeatherConditions open-source [code sample](#).

Next steps

If this FAQ doesn't answer your question, you can contact us through the following channels (in escalating order):

- The comments section of this article.
- [MSFT Q&A page for Azure Maps](#).
- Microsoft Support. To create a new support request, in the [Azure portal](#), on the Help tab, select the Help + support button, and then select **New support request**.
- [Azure Maps UserVoice](#) to submit feature requests.

Learn how to request real-time and forecasted weather data using Azure Maps Weather services:

[Request Real-time weather data](#)

Azure Maps Weather services concepts article:

[Weather services concepts](#)

Explore the Azure Maps Weather services API documentation:

[Azure Maps Weather services](#)

Create your Azure Maps account using an ARM template

6/9/2021 • 2 minutes to read • [Edit Online](#)

You can create your Azure Maps account using an Azure Resource Manager (ARM) template. After you have an account, you can implement the APIs in your website or mobile application.

An [ARM template](#) is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.



Prerequisites

To complete this article:

- If you don't have an Azure subscription, create a [free account](#) before you begin.

Review the template

The template used in this quickstart is from [Azure Quickstart Templates](#).

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "accountName": {
            "type": "string",
            "defaultValue": "[uniqueString(resourceGroup().id)]",
            "metadata": {
                "description": "The name for your Azure Maps account. This value must be globally unique."
            }
        },
        "pricingTier": {
            "type": "string",
            "allowedValues": [
                "S0",
                "S1",
                "G2"
            ],
            "defaultValue": "G2",
            "metadata": {
                "description": "The pricing tier for the account. Use S0 for small-scale development. Use S1 or G2 for large-scale applications."
            }
        },
        "kind": {
            "type": "string",
            "allowedValues": [
                "Gen1",
                "Gen2"
            ],
            "defaultValue": "Gen2",
            "metadata": {
                "description": "The pricing tier for the account. Use Gen1 for small-scale development. Use Gen2 for large-scale applications."
            }
        }
    },
    "resources": [
        {
            "name": "[parameters('accountName')]",
            "type": "Microsoft.Maps/accounts",
            "apiVersion": "2021-02-01",
            "location": "global",
            "sku": {
                "name": "[parameters('pricingTier')]"
            },
            "kind": "[parameters('kind')]"
        }
    ]
}
```

The Azure Maps account resource is defined in this template:

- **Microsoft.Maps/accounts**: create an Azure Maps account.

Deploy the template

1. Select the following image to sign in to Azure and open a template. The template creates an Azure Maps account.



2. Select or enter the following values.

Microsoft Azure Search resources, services, and docs (G+) ≡ ? ?

Home > Deploy Azure Maps

Azure quickstart template

Basics Review + create

Template

101-maps-create 1 resource Edit template Edit parameters

Deployment scope

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * <Azure subscription name> Create new

Resource group * West US 2 Create new

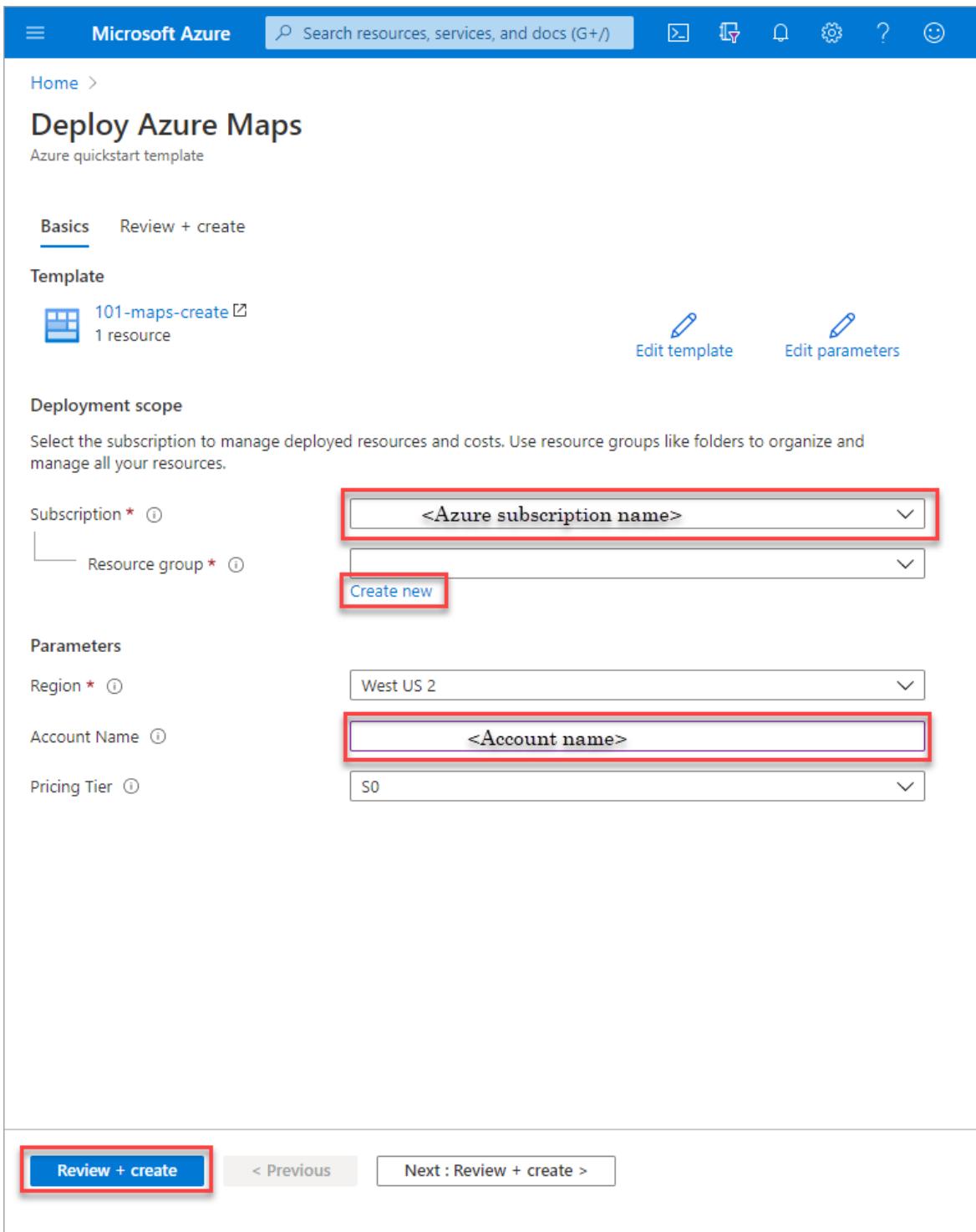
Parameters

Region * West US 2

Account Name <Account name>

Pricing Tier S0

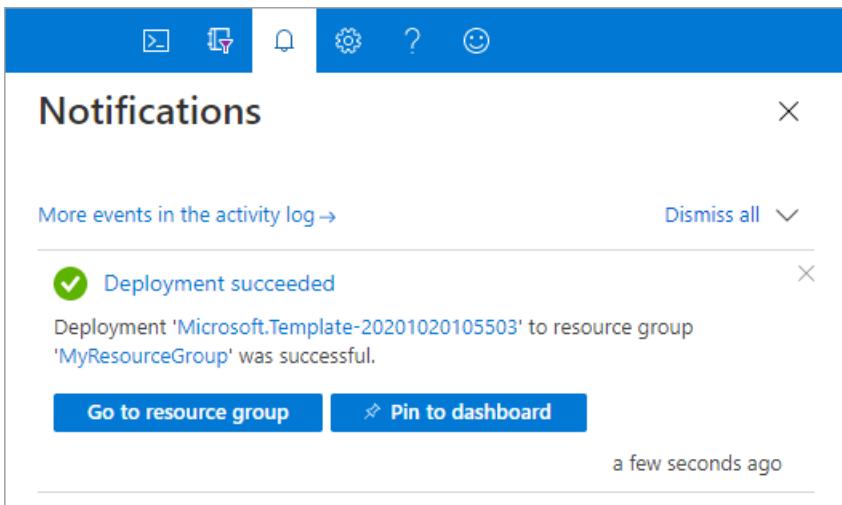
Review + create < Previous Next : Review + create >



Unless it's specified, use the default value to create your Azure Maps account.

- **Subscription:** select an Azure subscription.
- **Resource group:** select **Create new**, enter a unique name for the resource group, and then click **OK**.
- **Location:** select a location. For example, **West US 2**.
- **Account Name:** enter a name for your Azure Maps account, which must be globally unique.
- **Pricing Tier:** select the appropriate pricing tier, the default value for the template is **S0**.

3. Select **Review + create**.
4. Confirm your settings on the review page and click **Create**. After your Azure Maps has been deployed successfully, you get a notification:



The Azure portal is used to deploy your template. You can also use the Azure PowerShell, Azure CLI, and REST API. To learn other deployment methods, see [Deploy templates](#).

Review deployed resources

You can use the Azure portal to check your Azure Maps account and view your keys. You can also use the following Azure CLI script to list your account keys.

```
az maps account keys list --name MyMapsAccount --resource-group MyResourceGroup
```

Clean up resources

When no longer needed, delete the resource group, which also deletes the Azure Maps account. To delete the resource group by using Azure CLI:

```
az group delete --name MyResourceGroup
```

Next steps

To learn more about Azure Maps and Azure Resource Manager, continue on to the articles below.

- Create an Azure Maps [demo application](#)
- Learn more about [ARM templates](#)

Manage your Azure Maps account

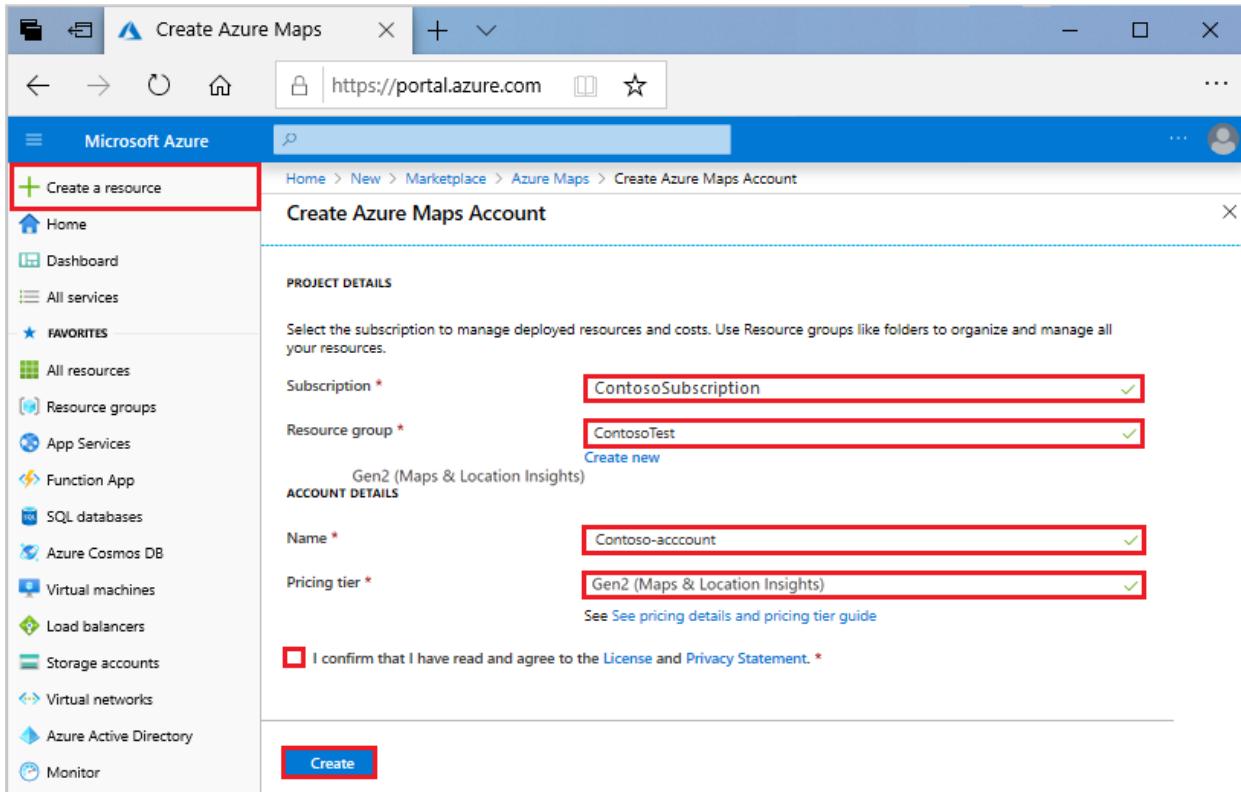
4/30/2021 • 2 minutes to read • [Edit Online](#)

You can manage your Azure Maps account through the Azure portal. After you have an account, you can implement the APIs in your website or mobile application.

If you don't have an Azure subscription, create a [free account](#) before you begin.

Create a new account

1. Sign in to the [Azure portal](#).
2. Select **Create a resource** in the upper-left corner of the Azure portal.
3. Search for and select **Maps**. Then select **Create**.
4. Enter the information for your new account.



Delete an account

You can delete an account from the Azure portal. Navigate to the account overview page and select **Delete**.

The screenshot shows the Microsoft Azure portal interface for managing an Azure Maps account named 'Contoso-account'. On the left, there's a navigation sidebar with various options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings (Authentication, Pricing Tier, Properties, Locks, Export template), Help (Getting Started, About), Monitoring (Alerts, Metrics, Diagnostic settings), Support + troubleshooting, and New support request. The main content area displays resource details: Resource group (change) <resource group>, Location <location>, Subscription (change) <subscription>, Subscription ID <subscription id>, and Tags (change) <tags>. It also shows the Pricing Tier (SKU) as Gen2 (Maps & Location Insights) and Client ID as <client id>. Below this, there are three line charts: 'Total Requests', 'Total Errors', and 'Availability'. Each chart has a Y-axis from 0 to 100 and an X-axis from 12 PM to UTC-01:00. The 'Total Requests' chart shows a single data series for 'contoso-account' starting at 100. The 'Total Errors' chart shows a single data series for 'contoso-account' starting at 0. The 'Availability' chart shows a single data series for 'contoso-account' starting at 100. At the top right, there are buttons for Move, Delete (which is highlighted with a red box), Refresh, and a search bar.

You then see a confirmation page. You can confirm the deletion of your account by typing its name.

Next steps

Set up authentication with Azure Maps and learn how to get an Azure Maps subscription key:

[Manage authentication](#)

Learn how to manage an Azure Maps account pricing tier:

[Manage a pricing tier](#)

Learn how to see the API usage metrics for your Azure Maps account:

[View usage metrics](#)

Manage authentication in Azure Maps

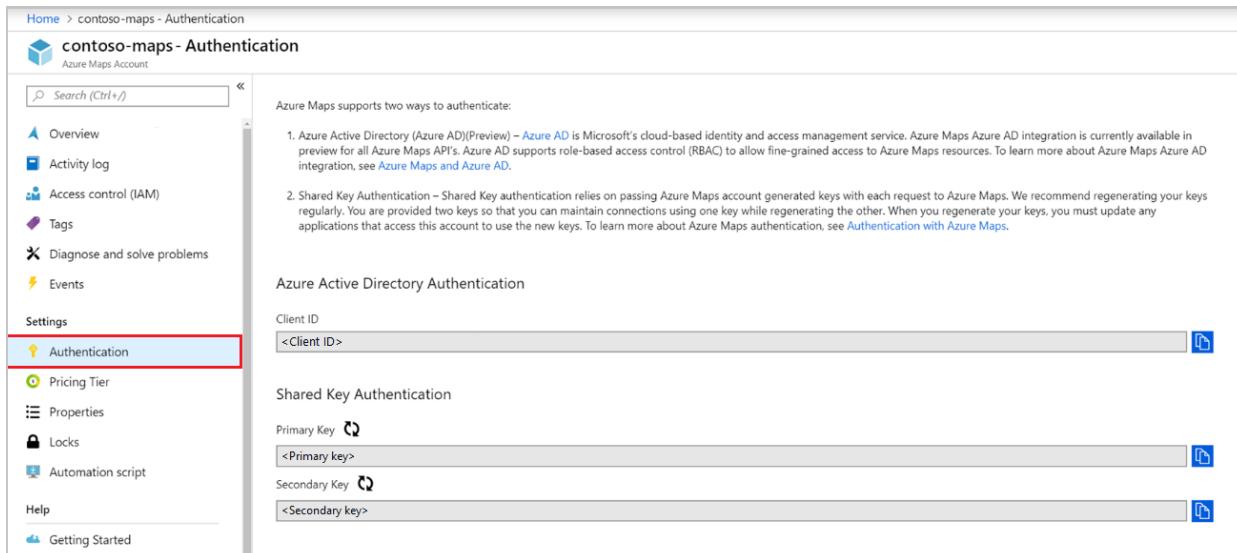
3/23/2021 • 3 minutes to read • [Edit Online](#)

After you create an Azure Maps account, a client ID and keys are created to support Azure Active Directory (Azure AD) authentication and Shared Key authentication.

View authentication details

After you create an Azure Maps account, the primary and secondary keys are generated. We recommend that you use a primary key as a subscription key when you [use Shared Key authentication to call Azure Maps](#). You can use a secondary key in scenarios such as rolling key changes. For more information, see [Authentication in Azure Maps](#).

You can view your authentication details in the Azure portal. There, in your account, on the **Settings** menu, select **Authentication**.



Discover category and scenario

Depending on application needs there are specific pathways to securing the application. Azure AD defines categories to support a wide range of authentication flows. See [application categories](#) to understand which category the application fits.

NOTE

Even if you use shared key authentication, understanding categories and scenarios helps you to secure the application.

Determine authentication and authorization

The following table outlines common authentication and authorization scenarios in Azure Maps. The table provides a comparison of the types of protection each scenario offers.

IMPORTANT

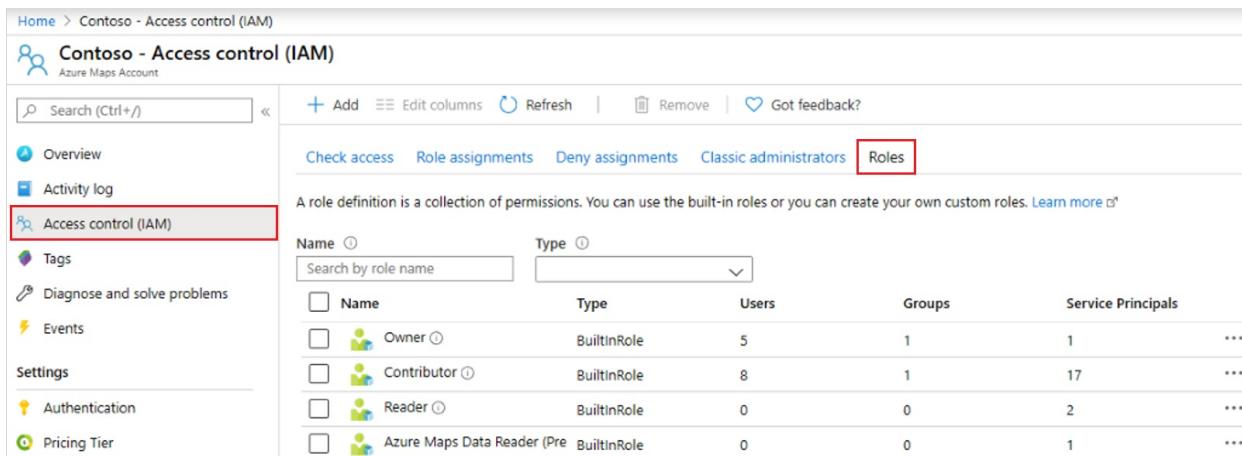
Microsoft recommends implementing Azure Active Directory (Azure AD) with Azure role-based access control (Azure RBAC) for production applications.

SCENARIO	AUTHENTICATION	AUTHORIZATION	DEVELOPMENT EFFORT	OPERATIONAL EFFORT
Trusted daemon / non-interactive client application	Shared Key	N/A	Medium	High
Trusted daemon / non-interactive client application	Azure AD	High	Low	Medium
Web single page application with interactive single-sign-on	Azure AD	High	Medium	Medium
Web single page application with non-interactive sign-on	Azure AD	High	Medium	Medium
Web application with interactive single-sign-on	Azure AD	High	High	Medium
IoT device / input constrained device	Azure AD	High	Medium	Medium

The links in the table take you to detailed configuration information for each scenario.

View role definitions

To view Azure roles that are available for Azure Maps, go to **Access control (IAM)**. Select **Roles**, and then search for roles that begin with *Azure Maps*. These Azure Maps roles are the roles that you can grant access to.

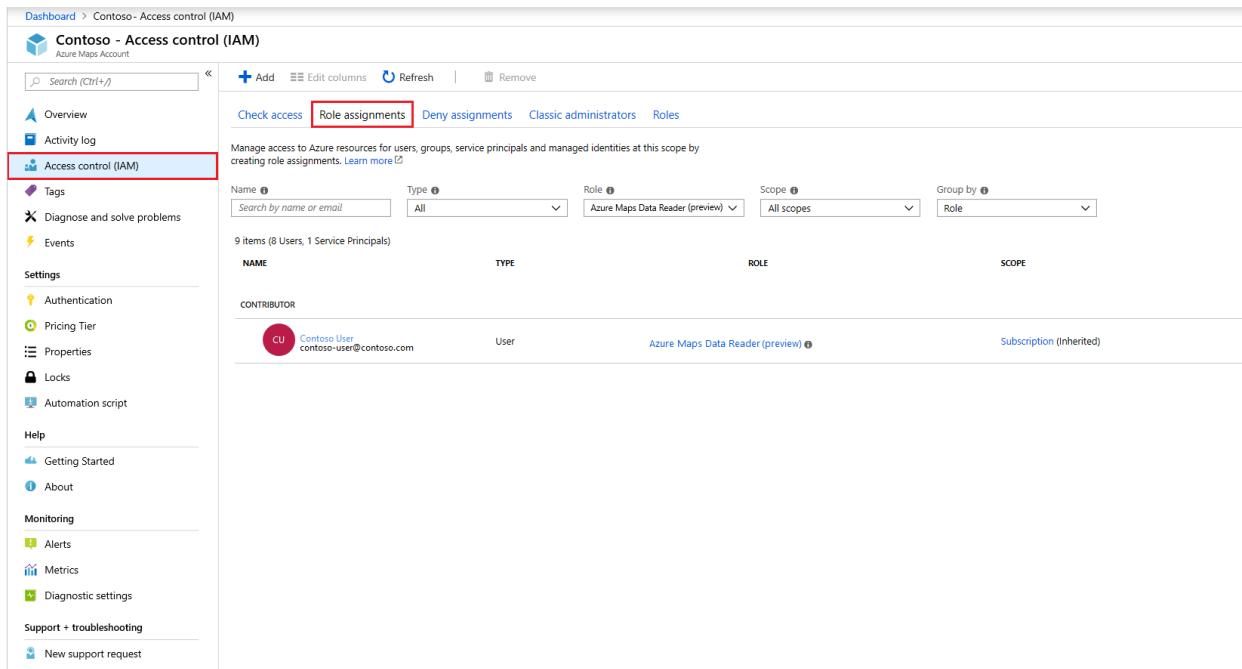


Name	Type	Users	Groups	Service Principals
Owner	BuiltInRole	5	1	1
Contributor	BuiltInRole	8	1	17
Reader	BuiltInRole	0	0	2
Azure Maps Data Reader (Pre)	BuiltInRole	0	0	1

View role assignments

To view users and apps that have been granted access for Azure Maps, go to **Access Control (IAM)**. There,

select **Role assignments**, and then filter by Azure Maps.



The screenshot shows the Azure portal's Access control (IAM) blade for the Contoso account. The 'Role assignments' tab is active. On the left, a sidebar lists various management options like Tags, Diagnose and solve problems, Events, Settings, Help, Monitoring, Support + troubleshooting, and Help. The 'Access control (IAM)' option is highlighted with a red box. The main area displays a table of role assignments. The table has columns for NAME, TYPE, ROLE, and SCOPE. One entry is shown: 'contoso-user@contoso.com' (User type), 'Azure Maps Data Reader (preview)' (Role), and 'All scopes' (Scope). The 'Scope' dropdown is also highlighted with a red box. The 'Group by' dropdown is set to 'Role'.

Request tokens for Azure Maps

Request a token from the Azure AD token endpoint. In your Azure AD request, use the following details:

AZURE ENVIRONMENT	AZURE AD TOKEN ENDPOINT	AZURE RESOURCE ID
Azure public cloud	https://login.microsoftonline.com	https://atlas.microsoft.com/
Azure Government cloud	https://login.microsoftonline.us	https://atlas.microsoft.com/

For more information about requesting access tokens from Azure AD for users and service principals, see [Authentication scenarios for Azure AD](#) and view specific scenarios in the table of [Scenarios](#).

Manage and rotate shared keys

Your Azure Maps subscription keys are similar to a root password for your Azure Maps account. Always be careful to protect your subscription keys. Use Azure Key Vault to manage and rotate your keys securely. Avoid distributing access keys to other users, hard-coding them, or saving them anywhere in plain text that is accessible to others. Rotate your keys if you believe they may have been compromised.

NOTE

Microsoft recommends using Azure Active Directory (Azure AD) to authorize requests if possible, instead of Shared Key. Azure AD provides superior security and ease of use over Shared Key.

Manually rotate subscription keys

Microsoft recommends that you rotate your subscription keys periodically to help keep your Azure Maps account secure. If possible, use Azure Key Vault to manage your access keys. If you are not using Key Vault, you will need to rotate your keys manually.

Two subscription keys are assigned so that you can rotate your keys. Having two keys ensures that your application maintains access to Azure Maps throughout the process.

To rotate your Azure Maps subscription keys in the Azure portal:

1. Update your application code to reference the secondary key for the Azure Maps account and deploy.
2. Navigate to your Azure Maps account in the [Azure portal](#).
3. Under **Settings**, select **Authentication**.
4. To regenerate the primary key for your Azure Maps account, select the **Regenerate** button next to the primary key.
5. Update your application code to reference the new primary key and deploy.
6. Regenerate the secondary key in the same manner.

WARNING

Microsoft recommends using only one of the keys in all of your applications at the same time. If you use Key 1 in some places and Key 2 in others, you will not be able to rotate your keys without some applications losing access.

Next steps

For more information, see [Azure AD and Azure Maps Web SDK](#).

Find the API usage metrics for your Azure Maps account:

[View usage metrics](#)

Explore samples that show how to integrate Azure AD with Azure Maps:

[Azure AD authentication samples](#)

Secure a daemon application

11/2/2020 • 6 minutes to read • [Edit Online](#)

The following guide is for background processes, timers, and jobs which are hosted in a trusted and secured environment. Examples include Azure Web Jobs, Azure Function Apps, Windows Services, and any other reliable background service.

TIP

Microsoft recommends implementing Azure Active Directory (Azure AD) and Azure role-based access control (Azure RBAC) for production applications. For an overview of concepts, see [Azure Maps Authentication](#).

You can view the Azure Maps account authentication details in the Azure portal. There, in your account, on the **Settings** menu, select **Authentication**.

The screenshot shows the Azure portal interface for managing an Azure Maps account named "contoso-maps". The left sidebar lists various account settings like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings, and Authentication. The "Authentication" item is highlighted with a red border. The main content area displays two authentication methods: Azure Active Directory Authentication and Shared Key Authentication. Under Azure Active Directory Authentication, there is a "Client ID" input field containing "<Client ID>". Under Shared Key Authentication, there are three input fields: "Primary Key" containing "<Primary key>", "Secondary Key" containing "<Secondary key>", and another "Secondary Key" field below it. Each input field has a copy icon (a blue square with a white 'C') to its right.

Once an Azure Maps account is created, the Azure Maps `x-ms-client-id` value is present in the Azure portal authentication details page. This value represents the account which will be used for REST API requests. This value should be stored in application configuration and retrieved prior to making http requests when using Azure AD authentication with Azure Maps.

Scenario: Shared key authentication

After you create an Azure Maps account, the primary and secondary keys are generated. We recommend that you use the primary key as the subscription key when you [use shared key authentication to call Azure Maps](#). You can use a secondary key in scenarios such as rolling key changes. For more information, see [Authentication in Azure Maps](#).

Securely store shared key

The primary and secondary key allow authorization to all APIs for the Maps account. Applications should store the keys in a secure store such as Azure Key Vault. The application must retrieve the shared key as a Azure Key Vault secret to avoid storing the shared key in plain text in application configuration. To understand how to configure an Azure Key Vault, see [Azure Key Vault developer guide](#).

The following steps outline this process:

1. Create an Azure Key Vault.
2. Create an Azure AD service principal by creating an App registration or managed identity, the created principal is responsible to access Azure Key Vault.
3. Assign the service principal access to Azure Key secrets `Get` permission.
4. Temporarily assign access to secrets `Set` permission for you as the developer.
5. Set the shared key in the Key Vault secrets and reference the secret ID as configuration for the daemon application and remove your secrets `Set` permission.
6. Implement Azure AD authentication in the daemon application to retrieve the shared key secret from Azure Key Vault.
7. Create Azure Maps REST API request with shared key.

TIP

If the app is hosted in Azure environment, you should implement a Managed Identity to reduce the cost and complexity of managing a secret to authenticate to Azure Key Vault. See the following Azure Key Vault [tutorial to connect via managed identity](#).

The daemon application is responsible for retrieving the shared key from a secure storage. The implementation with Azure Key Vault requires authentication through Azure AD to access the secret. Instead, we encourage direct Azure AD authentication to Azure Maps as a result of the additional complexity and operational requirements of using shared key authentication.

IMPORTANT

To simplify key regeneration, we recommend applications use one key at a time. Applications can then regenerate the unused key and deploy the new regenerated key to a secured secret store such as Azure Key Vault.

Scenario: Azure AD role-based access control

Once an Azure Maps account is created, the Azure Maps `x-ms-client-id` value is present in the Azure portal authentication details page. This value represents the account which will be used for REST API requests. This value should be stored in application configuration and retrieved prior to making HTTP requests. The objective of the scenario is to enable the daemon application to authenticate to Azure AD and call Azure Maps REST APIs.

TIP

We recommend hosting on Azure Virtual Machines, Virtual Machine Scale Sets, or App Services to enable benefits of Managed Identity components.

Daemon hosted on Azure resources

When running on Azure resources, configure Azure managed identities to enable low cost, minimal credential management effort.

See [Overview of Managed Identities](#) to enable the application access to a Managed Identity.

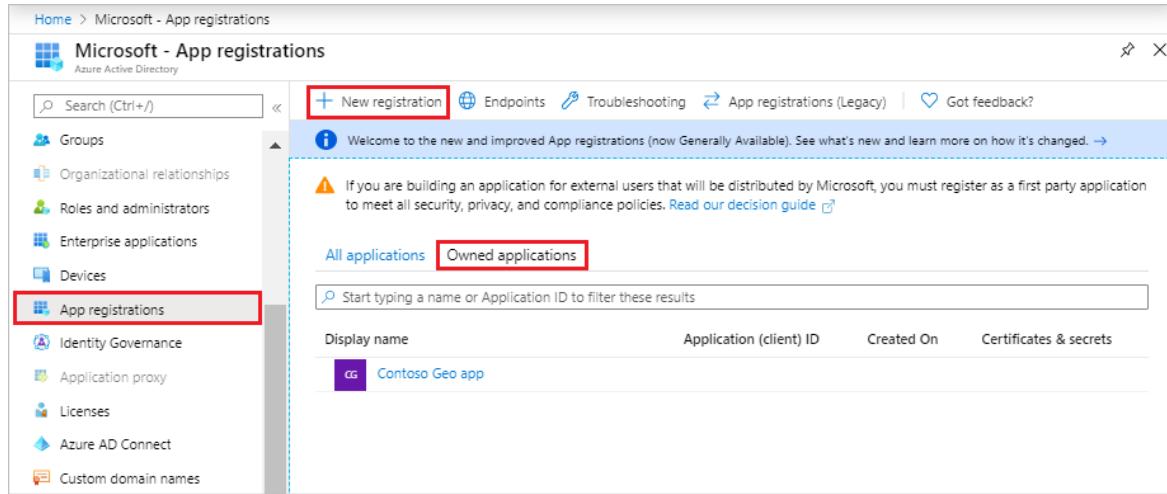
Managed Identity benefits:

- Azure system managed X509 certificate public key cryptography authentication.
- Azure AD security with X509 certificates instead of client secrets.
- Azure manages and renews all certificates associated with the Managed Identity resource.
- Simplified credential operational management by removing any need for a secured secret store service like Azure Key Vault.

Daemon hosted on non-Azure resources

When running on a non-Azure environment Managed Identities are not available. Therefore you must configure a service principal through an Azure AD application registration for the daemon application.

1. In the Azure portal, in the list of Azure services, select **Azure Active Directory > App registrations > New registration**.



The screenshot shows the 'Microsoft - App registrations' page in the Azure portal. On the left, there's a sidebar with various options like Groups, Organizational relationships, Roles and administrators, Enterprise applications, Devices, and App registrations. The 'App registrations' option is highlighted with a red box. At the top right, there are buttons for 'Endpoints', 'Troubleshooting', 'App registrations (Legacy)', and 'Got feedback?'. Below these, a welcome message says: 'Welcome to the new and improved App registrations (now Generally Available). See what's new and learn more on how it's changed.' A note below it states: 'If you are building an application for external users that will be distributed by Microsoft, you must register as a first party application to meet all security, privacy, and compliance policies. [Read our decision guide](#)' with a small blue arrow icon. The main area has tabs for 'All applications' and 'Owned applications', with 'Owned applications' selected and highlighted with a red box. There's a search bar with placeholder text 'Start typing a name or Application ID to filter these results'. A table below lists applications with columns for 'Display name', 'Application (client) ID', 'Created On', and 'Certificates & secrets'. One row is visible, showing 'Contoso Geo app'.

2. If you've already registered your app, then continue to the next step. If you haven't registered your app, then enter a **Name**, choose a **Support account type**, and then select **Register**.

Register an application

⚠️ If you are building an application for external users that will be distributed by Microsoft, you must register as a first party application to meet all security, privacy, and compliance policies. [Read our decision guide ↗](#)

* Name

The user-facing display name for this application (this can be changed later).

✓

Supported account types

Who can use this application or access this API?

- Accounts in this organizational directory only (Microsoft only - Single tenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

✓

✓

[By proceeding, you agree to the Microsoft Platform Policies ↗](#)

Register

- To assign delegated API permissions to Azure Maps, go to the application. Then under **App registrations**, select **API permissions** > **Add a permission**. Under **APIs my organization uses**, search for and select **Azure Maps**.

Request API permissions

Select an API

Microsoft APIs	APIs my organization uses	My APIs
--------------------------------	---	-------------------------

Apps in your directory that expose APIs are shown below

<input type="checkbox"/> Azure Maps

Name

Azure Maps

- Select the check box next to **Access Azure Maps**, and then select **Add permissions**.

Request API permissions

[All APIs](#)

Azure Maps

<https://atlas.microsoft.com/>

What type of permissions does your application require?

Delegated permissions

Your application needs to access the API as the signed-in user.

Application permissions

Your application runs as a background service or daemon without a signed-in user.

Select permissions

[expand all](#)

Type to search

Permission Admin consent req...

user_impersonation
Access Azure Maps [\(i\)](#)

[Add permissions](#)

[Discard](#)

5. Complete the following steps to create a client secret or configure certificate.

- If your application uses server or application authentication, then on your app registration page, go to **Certificates & secrets**. Then either upload a public key certificate or create a password by selecting **New client secret**.

The screenshot shows the 'Certificates & secrets' section of the Azure App Registration. On the left, there's a sidebar with links like Overview, Quickstart, Manage, and Certificates & secrets (which is highlighted). The main area has two tabs: 'Certificates' and 'Client secrets'. Under 'Certificates', it says 'No certificates have been added for this application.' and has a 'Upload certificate' button. Under 'Client secrets', it says 'A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.' and has a red-bordered 'New client secret' button. There's also a table for managing client secrets with columns: Description, Expires, and Value.

- After you select Add, copy the secret and store it securely in a service such as Azure Key Vault. Review [Azure Key Vault Developer Guide](#) to securely store the certificate or secret. You'll use this secret to get tokens from Azure AD.

The screenshot shows the 'Certificates & secrets' page for the 'Contoso Geo app'. On the left, there's a sidebar with links like Overview, Quickstart, Manage, Branding, Authentication, Certificates & secrets (which is highlighted with a red box), Token configuration (preview), and API permissions. The main area has a search bar, an 'Add a client secret' button, and a form for creating a new secret. The 'Description' field contains 'Contoso Geo app password', the 'Expires' section has 'In 1 year' selected (radio button is checked), and there are 'Cancel' and 'Add' buttons at the bottom.

Grant role-based access for the daemon application to Azure Maps

You grant *Azure role-based access control (Azure RBAC)* by assigning either the created Managed Identity or the service principal to one or more Azure Maps role definitions. To view Azure role definitions that are available for Azure Maps, go to **Access control (IAM)**. Select **Roles**, and then search for roles that begin with *Azure Maps*. These Azure Maps roles are the roles that you can grant access to.

The screenshot shows the 'Access control (IAM)' page for the 'Contoso' account. The left sidebar includes links for Overview, Activity log, Access control (IAM) (which is highlighted with a red box), Tags, Diagnose and solve problems, Events, Settings, Authentication, and Pricing Tier. The main area has tabs for Check access, Role assignments, Deny assignments, Classic administrators, and Roles (which is highlighted with a red box). Below is a table of role definitions:

Name	Type	Users	Groups	Service Principals
<input type="checkbox"/> Name	Type			
<input type="checkbox"/> Owner	BuiltInRole	5	1	1
<input type="checkbox"/> Contributor	BuiltInRole	8	1	17
<input type="checkbox"/> Reader	BuiltInRole	0	0	2
<input type="checkbox"/> Azure Maps Data Reader (Pre)	BuiltInRole	0	0	1

1. Go to your Azure Maps Account. Select **Access control (IAM) > Role assignments**.

The screenshot shows the 'Access control (IAM)' page for the 'Contoso' account. The left sidebar includes links for Overview, Activity log, and Access control (IAM) (which is highlighted with a red box). The main area has tabs for Check access, Role assignments (which is highlighted with a red box), Deny assignments, Classic administrators, and Roles. Below is a summary message: 'Manage access to Azure resources for users, groups, service principals and managed identities at this scope by creating role assignments.' It also displays 'Number of role assignments for this subscription'.

2. On the **Role assignments** tab, Add a role assignment.

[Add](#) [Edit columns](#) [Refresh](#) | [Remove](#) | [Got feedback?](#)

Check access [Role assignments](#) Deny assignments Classic administrators Roles

Manage access to Azure resources for users, groups, service principals and managed identities at this scope by creating role assignments. [Learn more](#)

Number of role assignments for this subscription [\(i\)](#)

35 2000

Name (i)	Type (i)	Role (i)
Contoso	Apps	5 selected

Scope [\(i\)](#) Group by [\(i\)](#)
All scopes Role

i Showing a filtered set of results. Total number of role assignments: 10

1 items (1 Service Principals)

<input type="checkbox"/> Name	Type	Role	Scope
Azure Maps Data Reader			
<input type="checkbox"/>  Contoso Geo app	App	Azure Maps Data Reader (i)	This resource

3. Select a built-in Azure Maps role definition such as **Azure Maps Data Reader** or **Azure Maps Data Contributor**. Under **Assign access to**, select **Azure AD user, group, or service principal** or **Managed Identity with User assigned managed identity / System assigned Managed identity**. Select the principal. Then select **Save**.

Add role assignment

[X](#)

Role [\(i\)](#)
(i)

Assign access to [\(i\)](#)

Select [\(i\)](#)

No users, groups, or service principals found.

Selected members:

 Contoso Geo app	Remove
---	------------------------

[Save](#) [Discard](#)

4. You can confirm the role assignment was applied on the role assignment tab.

Request token with Managed Identity

Once a managed identity is configured for the hosting resource, use Azure SDK or REST API to acquire a token

for Azure Maps, see details on [Acquire an access token](#). Following the guide, the expectation is that an access token will be returned which can be used on REST API requests.

Request token with application registration

After you register your app and associate it with Azure Maps, you can request access tokens.

- Azure AD resource ID `https://atlas.microsoft.com/`
- Azure AD App ID
- Azure AD Tenant ID
- Azure AD App registration client secret

Request:

```
POST /<Azure AD Tenant ID>/oauth2/token HTTP/1.1
Host: login.microsoftonline.com
Content-Type: application/x-www-form-urlencoded

client_id=<Azure AD App ID>&resource=https://atlas.microsoft.com/&client_secret=<client
secret>&grant_type=client_credentials
```

Response:

```
{
  "token_type": "Bearer",
  "expires_in": "...",
  "ext_expires_in": "...",
  "expires_on": "...",
  "not_before": "...",
  "resource": "https://atlas.microsoft.com/",
  "access_token": "ey...gw"
}
```

See [Authentication scenarios for Azure AD](#), for more detailed examples.

Next steps

Find the API usage metrics for your Azure Maps account:

[View usage metrics](#)

Explore samples that show how to integrate Azure AD with Azure Maps:

[Azure Maps samples](#)

Secure a single page application with user sign-in

11/2/2020 • 3 minutes to read • [Edit Online](#)

The following guide pertains to an application which is hosted on a content server or has minimal web server dependencies. The application provides protected resources secured only to Azure AD users. The objective of the scenario is to enable the web application to authenticate to Azure AD and call Azure Maps REST APIs on behalf of the user.

You can view the Azure Maps account authentication details in the Azure portal. There, in your account, on the **Settings** menu, select **Authentication**.

The screenshot shows the Azure portal interface for a 'contoso-maps' account. The left sidebar lists various settings like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, and Authentication. The 'Authentication' item is highlighted with a red box. The main content area displays information about Azure Maps authentication methods:

- Azure Active Directory (Azure AD)(Preview)**: Describes Azure AD as Microsoft's cloud-based identity and access management service. It mentions that Azure Maps Azure AD integration is currently available in preview for all Azure Maps API's. It supports role-based access control (RBAC) to allow fine-grained access to Azure Maps resources. A link to 'Azure Maps and Azure AD' is provided.
- Shared Key Authentication**: Describes Shared Key authentication as relying on passing Azure Maps account generated keys with each request to Azure Maps. It recommends regenerating your keys regularly. It notes that two keys are provided so that you can maintain connections using one key while regenerating the other. A link to 'Authentication with Azure Maps' is provided.

Below this, there are sections for 'Azure Active Directory Authentication' (Client ID field) and 'Shared Key Authentication' (Primary Key and Secondary Key fields).

Once an Azure Maps account is created, the Azure Maps `x-ms-client-id` value is present in the Azure portal authentication details page. This value represents the account which will be used for REST API requests. This value should be stored in application configuration and retrieved prior to making http requests when using Azure AD authentication with Azure Maps.

Create an application registration in Azure AD

Create the web application in Azure AD for users to sign in. The web application delegates user access to Azure Maps REST APIs.

1. In the Azure portal, in the list of Azure services, select **Azure Active Directory > App registrations > New registration**.

Welcome to the new and improved App registrations (now Generally Available). See what's new and learn more on how it's changed. [Read our decision guide](#)

All applications **Owned applications**

Start typing a name or Application ID to filter these results

Display name	Application (client) ID	Created On	Certificates & secrets
Contoso Geo app			

2. Enter a **Name**, choose a **Support account type**, provide a redirect URI which will represent the url which Azure AD will issue the token and is the url where the map control is hosted. For a detailed sample please see [Azure Maps Azure AD samples](#). Then select **Register**.
3. To assign delegated API permissions to Azure Maps, go to the application. Then under **App registrations**, select **API permissions > Add a permission**. Under **APIs my organization uses**, search for and select **Azure Maps**.

Name
Azure Maps

4. Select the check box next to **Access Azure Maps**, and then select **Add permissions**.

Request API permissions

X

[All APIs](#)

Azure Maps

<https://atlas.microsoft.com/>

What type of permissions does your application require?

Delegated permissions

Your application needs to access the API as the signed-in user.

Application permissions

Your application runs as a background service or daemon without a signed-in user.

Select permissions

[expand all](#)

Type to search

Permission	Admin consent req...
------------	----------------------



user_impersonation
Access Azure Maps [\(i\)](#)

[Add permissions](#)

[Discard](#)

5. Enable `oauth2AllowImplicitFlow`. To enable it, in the **Manifest** section of your app registration, set `oauth2AllowImplicitFlow` to `true`.
6. Copy the Azure AD app ID and the Azure AD tenant ID from the app registration to use in the Web SDK. Add the Azure AD app registration details and the `x-ms-client-id` from the Azure Map account to the Web SDK.

```
<link rel="stylesheet"
      href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css" />
<script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js" />
<script>
    var map = new atlas.Map("map", {
        center: [-122.33, 47.64],
        zoom: 12,
        language: "en-US",
        authOptions: {
            authType: "aad",
            clientId: "<insert>", // azure map account client id
            aadAppId: "<insert>", // azure ad app registration id
            aadTenant: "<insert>", // azure ad tenant id
            aadInstance: "https://login.microsoftonline.com/"
        }
    });
</script>
```

7. Configure Azure role-based access control (Azure RBAC) for users or groups. See the [following sections to enable Azure RBAC](#).

Grant role-based access for users to Azure Maps

You grant *Azure role-based access control (Azure RBAC)* by assigning either an Azure AD group or security principals to one or more Azure Maps role definitions. To view Azure role definitions that are available for Azure Maps, go to **Access control (IAM)**. Select **Roles**, and then search for roles that begin with *Azure Maps*.

- To efficiently manage a large amount of users' access to Azure Maps, see [Azure AD Groups](#).
- For users to be allowed to authenticate to the application, the users must be created in Azure AD. See [Add or Delete users using Azure AD](#).

Read more on [Azure AD](#) to effectively manage a directory for users.

1. Go to your Azure Maps Account. Select Access control (IAM) > Role assignment.

The screenshot shows the Azure portal interface for managing access control. At the top, it says 'Home > Contoso - Access control (IAM)'. Below that is a search bar and a toolbar with 'Add', 'Edit columns', 'Refresh', 'Remove', and 'Got feedback?'. Underneath is a navigation bar with 'Overview', 'Check access' (which is the active tab), 'Role assignments' (which is highlighted with a red border), 'Deny assignments', 'Classic administrators', and 'Roles'. A note below the navigation bar says 'Manage access to Azure resources for users, groups, service principals and managed identities at this scope by creating role assignments.' followed by a 'Learn more' link. At the bottom of the main area, there's a button 'Number of role assignments for this subscription' with a help icon.

2. On the **Role assignments** tab, under **Role**, select a built in Azure Maps role definition such as **Azure Maps Data Reader** or **Azure Maps Data Contributor**. Under **Assign access to**, select **Azure AD user, group, or service principal**. Select the principal by name. Then select **Save**.

- See details on [Assign Azure roles](#).

WARNING

Azure Maps built-in role definitions provide a very large authorization access to many Azure Maps REST APIs. To restrict APIs for users to a minimum, see [create a custom role definition and assign users](#) to the custom role definition. This will enable users to have the least privilege necessary for the application.

Next steps

Further understanding of single page application scenario:

[Single-page application](#)

Find the API usage metrics for your Azure Maps account:

[View usage metrics](#)

Explore samples that show how to integrate Azure AD with Azure Maps:

[Azure Maps Samples](#)

Secure a web application with user sign-in

11/2/2020 • 4 minutes to read • [Edit Online](#)

The following guide pertains to an application which is hosted on web servers, maintains multiple business scenarios, and deploys to web servers. The application has the requirement to provide protected resources secured only to Azure AD users. The objective of the scenario is to enable the web application to authenticate to Azure AD and call Azure Maps REST APIs on behalf of the user.

You can view the Azure Maps account authentication details in the Azure portal. There, in your account, on the **Settings** menu, select **Authentication**.

The screenshot shows the Azure portal interface for managing an Azure Maps account named "contoso-maps". The left sidebar lists various settings like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, and Authentication. The "Authentication" item is highlighted with a red box. The main content area displays information about authentication methods, specifically Azure Active Directory (Azure AD) and Shared Key Authentication. It includes fields for Client ID, Primary Key, and Secondary Key, each with a copy icon.

Once an Azure Maps account is created, the Azure Maps `x-ms-client-id` value is present in the Azure portal authentication details page. This value represents the account which will be used for REST API requests. This value should be stored in application configuration and retrieved prior to making http requests when using Azure AD authentication with Azure Maps.

Create an application registration in Azure AD

You must create the web application in Azure AD for users to sign in. This web application will then delegate user access to Azure Maps REST APIs.

1. In the Azure portal, in the list of Azure services, select **Azure Active Directory > App registrations > New registration**.

Home > Microsoft - App registrations

Microsoft - App registrations

New registration Endpoints Troubleshooting App registrations (Legacy) Got feedback?

Welcome to the new and improved App registrations (now Generally Available). See what's new and learn more on how it's changed. →

If you are building an application for external users that will be distributed by Microsoft, you must register as a first party application to meet all security, privacy, and compliance policies. [Read our decision guide](#)

All applications Owned applications

Start typing a name or Application ID to filter these results

Display name	Application (client) ID	Created On	Certificates & secrets
Contoso Geo app			

2. Enter a **Name**, choose a **Support account type**, provide a redirect URI which will represent the url which Azure AD will issue the token and is the url where the map control is hosted. For more details please see Azure AD [Scenario: Web app that signs in users](#). Complete the provided steps from the Azure AD scenario.
3. Once the application registration is complete, Confirm that application sign-in works for users. Once sign-in works, then the application can be granted delegated access to Azure Maps REST APIs.
4. To assign delegated API permissions to Azure Maps, go to the application. Then select **API permissions > Add a permission**. Under **APIs my organization uses**, search for and select **Azure Maps**.

Request API permissions

Select an API

Microsoft APIs APIs my organization uses My APIs

Apps in your directory that expose APIs are shown below

Azure Maps

Name
Azure Maps

5. Select the check box next to **Access Azure Maps**, and then select **Add permissions**.

Request API permissions

X

[All APIs](#)

Azure Maps

<https://atlas.microsoft.com/>

What type of permissions does your application require?

Delegated permissions

Your application needs to access the API as the signed-in user.

Application permissions

Your application runs as a background service or daemon without a signed-in user.

Select permissions

[expand all](#)

Type to search

Permission	Admin consent req...
------------	----------------------

<input checked="" type="checkbox"/> user_impersonation	Access Azure Maps (i)
--	---------------------------------------

[Add permissions](#)

[Discard](#)

6. Enable the web application to call Azure Maps REST APIs by configuring the app registration with an application secret, For detailed steps, see [A web app that calls web APIs: App registration](#). A secret is required to authenticate to Azure AD on-behalf of the user. The app registration certificate or secret should be stored in a secure store for the web application to retrieve to authenticate to Azure AD.
 - If the application already has configured an Azure AD app registration and a secret this step may be skipped.

TIP

If the application is hosted in an Azure environment, we recommend using [Managed identities for Azure resources](#) and an Azure Key Vault instance to access secrets by [acquiring an access token](#) for accessing Azure Key Vault secrets or certificates. To connect to Azure Key Vault to retrieve secrets, see [tutorial to connect through managed identity](#).

7. Implement a secure token endpoint for the Azure Maps Web SDK to access a token.
 - For a sample token controller, see [Azure Maps Azure AD Samples](#).
 - For a non-AspNetCore implementation or other, see [Acquire token for the app](#) from Azure AD documentation.
 - The secured token endpoint is responsible to return an access token for the authenticated and authorized user to call Azure Maps REST APIs.
8. Configure Azure role-based access control (Azure RBAC) for users or groups. See [grant role-based access for users](#).
9. Configure the web application page with the Azure Maps Web SDK to access the secure token endpoint.

```

var map = new atlas.Map("map", {
    center: [-122.33, 47.64],
    zoom: 12,
    language: "en-US",
    authOptions: {
        authType: "anonymous",
        clientId: "<insert>", // azure map account client id
        getToken: function (resolve, reject, map) {
            var xhttp = new XMLHttpRequest();
            xhttp.open("GET", "/api/token", true); // the url path maps to the token endpoint.
            xhttp.onreadystatechange = function () {
                if (this.readyState === 4 && this.status === 200) {
                    resolve(this.responseText);
                } else if (this.status !== 200) {
                    reject(this.responseText);
                }
            };
            xhttp.send();
        }
    }
});
map.events.add("tokenacquired", function () {
    console.log("token acquired");
});
map.events.add("error", function (err) {
    console.log(JSON.stringify(err.error));
});

```

Grant role-based access for users to Azure Maps

You grant *Azure role-based access control (Azure RBAC)* by assigning either an Azure AD group or security principals to one or more Azure Maps role definitions. To view Azure role definitions that are available for Azure Maps, go to **Access control (IAM)**. Select **Roles**, and then search for roles that begin with *Azure Maps*.

- To efficiently manage a large amount of users' access to Azure Maps, see [Azure AD Groups](#).
- For users to be allowed to authenticate to the application, the users must be created in Azure AD. See [Add or Delete users using Azure AD](#).

Read more on [Azure AD](#) to effectively manage a directory for users.

1. Go to your **Azure Maps Account**. Select **Access control (IAM)** > **Role assignment**.

The screenshot shows the Azure portal's 'Access control (IAM)' blade for a specific Azure Maps account. The top navigation bar shows 'Home > Contoso - Access control (IAM)'. The main area has a title 'Contoso - Access control (IAM)' and a subtitle 'Azure Maps Account'. Below this is a toolbar with 'Search (Ctrl+ /)', 'Add', 'Edit columns', 'Refresh', 'Remove', and 'Got feedback?'. A red box highlights the 'Access control (IAM)' link in the left sidebar. The main content area has tabs: 'Check access' (disabled), 'Role assignments' (selected and highlighted with a red border), 'Deny assignments', 'Classic administrators', and 'Roles'. Below the tabs is a message: 'Manage access to Azure resources for users, groups, service principals and managed identities at this scope by creating role assignments. Learn more'. At the bottom, it says 'Number of role assignments for this subscription' with a value of 0.

2. On the **Role assignments** tab, under **Role**, select a built in Azure Maps role definition such as **Azure Maps Data Reader** or **Azure Maps Data Contributor**. Under **Assign access to**, select **Azure AD user, group, or service principal**. Select the principal by name. Then select **Save**.

- See details on [Assign Azure roles](#).

WARNING

Azure Maps built-in role definitions provide a very large authorization access to many Azure Maps REST APIs. To restrict APIs for users to a minimum, see [create a custom role definition and assign users](#) to the custom role definition. This will enable users to have the least privilege necessary for the application.

Next steps

Further understanding of web application scenario:

[Scenario: Web app that signs in users](#)

Find the API usage metrics for your Azure Maps account:

[View usage metrics](#)

Explore samples that show how to integrate Azure AD with Azure Maps:

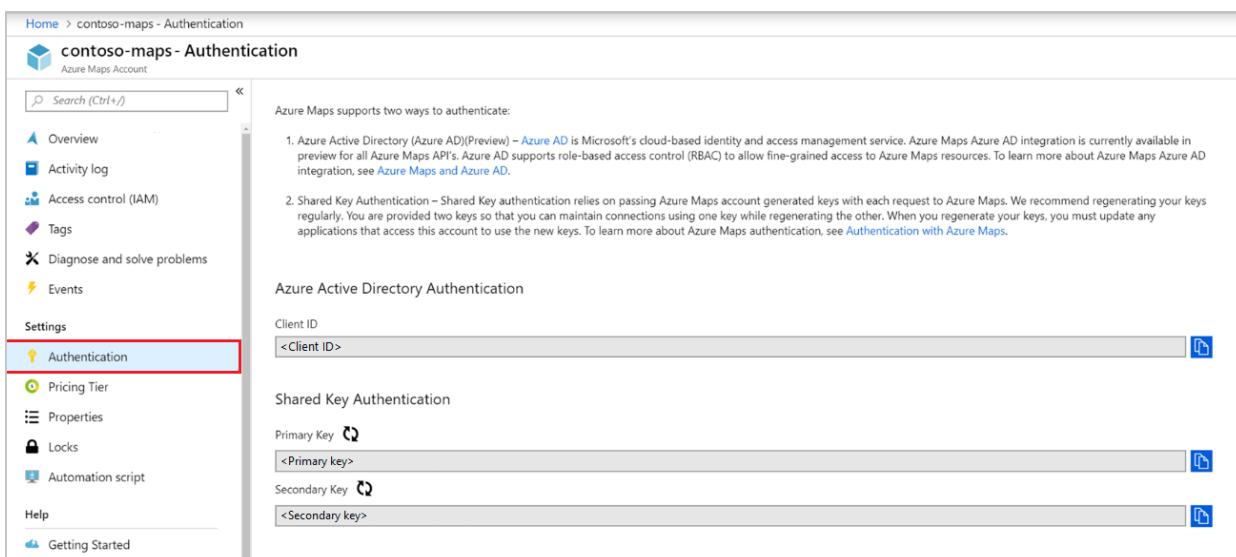
[Azure Maps Azure AD Web App Samples](#)

How to secure a single page application with non-interactive sign-in

3/5/2021 • 3 minutes to read • [Edit Online](#)

The following guide pertains to an application using Azure Active Directory (Azure AD) to provide an access token to Azure Maps applications when the user can't sign in to Azure AD. This flow requires hosting of a web service which must be secured to only be accessed by the single page web application. There are multiple implementations which can accomplish authentication to Azure AD. This guide leverages the product, Azure Function to acquire access tokens.

You can view the Azure Maps account authentication details in the Azure portal. There, in your account, on the **Settings** menu, select **Authentication**.



Once an Azure Maps account is created, the Azure Maps `x-ms-client-id` value is present in the Azure portal authentication details page. This value represents the account which will be used for REST API requests. This value should be stored in application configuration and retrieved prior to making http requests when using Azure AD authentication with Azure Maps.

TIP

Azure maps can support access tokens from user sign-on / interactive flows. Interactive flows enable a more restricted scope of access revocation and secret management.

Create Azure Function

Create a secured web service application which is responsible for authentication to Azure AD.

1. Create a function in the Azure portal. For more information, see [Create Azure Function](#).
2. Configure CORS policy on the Azure function to be accessible by the single page web application. This will secure browser clients to the allowed origins of your web application. See [Add CORS functionality](#).
3. [Add a system-assigned identity](#) on the Azure function to enable creation of a service principal to authenticate to Azure AD.

4. Grant role-based access for the system-assigned identity to the Azure Maps account. See [Grant role-based access](#) for details.

5. Write code for the Azure function to obtain Azure Maps access tokens using system-assigned identity with one of the supported mechanisms or the REST protocol. See [Obtain tokens for Azure resources](#)

A sample REST protocol example:

```
GET /MSI/token?resource=https://atlas.microsoft.com/&api-version=2019-08-01 HTTP/1.1
Host: localhost:4141
```

Sample response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access_token": "eyJ0eXAi...",
    "expires_on": "1586984735",
    "resource": "https://atlas.microsoft.com/",
    "token_type": "Bearer",
    "client_id": "..."
}
```

6. Configure security for the Azure function HttpTrigger

- [Create a function access key](#)
- [Secure HTTP endpoint](#) for the Azure function in production.

7. Configure web application Azure Maps Web SDK.

```
//URL to custom endpoint to fetch Access token
var url = 'https://<APP_NAME>.azurewebsites.net/api/<FUNCTION_NAME>?code=<API_KEY>';

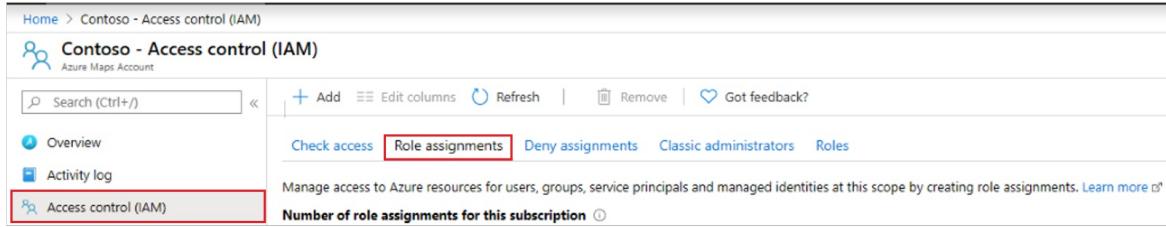
var map = new atlas.Map('myMap', {
    center: [-122.33, 47.6],
    zoom: 12,
    language: 'en-US',
    view: "Auto",
    authOptions: {
        authType: "anonymous",
        clientId: "<insert>", // azure map account client id
        getToken: function(resolve, reject, map) {
            fetch(url).then(function(response) {
                return response.text();
            }).then(function(token) {
                resolve(token);
            });
        }
    }
});

// use the following events to debug, you can remove them at any time.
map.events.add("tokenacquired", function () {
    console.log("token acquired");
});
map.events.add("error", function (err) {
    console.log(JSON.stringify(err.error));
});
```

Grant role-based access

You grant *Azure role-based access control (Azure RBAC)* access by assigning the system-assigned identity to one or more Azure role definitions. To view Azure role definitions that are available for Azure Maps, go to **Access control (IAM)**. Select **Roles**, and then search for roles that begin with *Azure Maps*.

1. Go to your Azure Maps Account. Select **Access control (IAM) > Role assignment**.



The screenshot shows the Azure portal's Access control (IAM) blade for a specific Azure Maps account named 'Contoso'. The 'Role assignments' tab is active. In the left sidebar, the 'Access control (IAM)' link is highlighted with a red box. Other options like 'Overview' and 'Activity log' are also visible.

2. On the **Role assignments** tab, under **Role**, select a built in Azure Maps role definition such as **Azure Maps Data Reader** or **Azure Maps Data Contributor**. Under **Assign access to**, select **Function App**. Select the principal by name. Then select **Save**.

- See details on [Assign Azure roles](#).

WARNING

Azure Maps built-in role definitions provide a very large authorization access to many Azure Maps REST APIs. To restrict APIs access to a minimum, see [create a custom role definition and assign the system-assigned identity](#) to the custom role definition. This will enable the least privilege necessary for the application to access Azure Maps.

Next steps

Further understanding of Single Page Application Scenario:

[Single-page application](#)

Find the API usage metrics for your Azure Maps account:

[View usage metrics](#)

Explore other samples that show how to integrate Azure AD with Azure Maps:

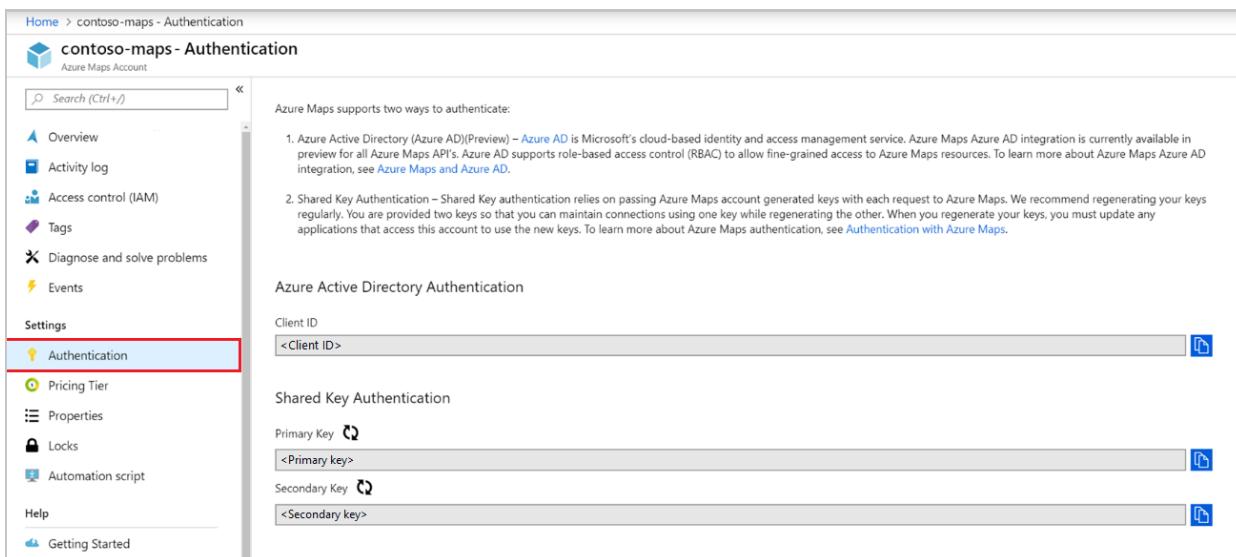
[Azure Maps Samples](#)

Secure an input constrained device with Azure AD and Azure Maps REST APIs

6/1/2021 • 3 minutes to read • [Edit Online](#)

This guide discusses how to secure public applications or devices that cannot securely store secrets or accept browser input. These types of applications fall under the category of IoT or internet of things. Some examples of these applications may include: Smart TV devices or sensor data emitting applications.

You can view the Azure Maps account authentication details in the Azure portal. There, in your account, on the **Settings** menu, select **Authentication**.



The screenshot shows the Azure portal's 'Authentication' settings for the 'contoso-maps' account. The 'Authentication' tab is selected in the left sidebar. The main pane displays two authentication methods: 'Azure Active Directory Authentication' and 'Shared Key Authentication'. Under 'Azure Active Directory Authentication', there is a 'Client ID' input field. Under 'Shared Key Authentication', there are 'Primary Key' and 'Secondary Key' input fields. Each input field has a copy icon (a blue square with a white 'C') next to it.

Once an Azure Maps account is created, the Azure Maps `x-ms-client-id` value is present in the Azure portal authentication details page. This value represents the account which will be used for REST API requests. This value should be stored in application configuration and retrieved prior to making http requests when using Azure AD authentication with Azure Maps.

Create an application registration in Azure AD

NOTE

- **Prerequisite Reading:** [Scenario: Desktop app that calls web APIs](#)
- The following scenario uses the device code flow, which does not involve a web browser to acquire a token.

Create the device based application in Azure AD to enable Azure AD sign in. This application will be granted access to Azure Maps REST APIs.

1. In the Azure portal, in the list of Azure services, select **Azure Active Directory > App registrations > New registration**.

Home > Microsoft - App registrations

Microsoft - App registrations

New registration Endpoints Troubleshooting App registrations (Legacy) Got feedback?

Welcome to the new and improved App registrations (now Generally Available). See what's new and learn more on how it's changed. →

If you are building an application for external users that will be distributed by Microsoft, you must register as a first party application to meet all security, privacy, and compliance policies. [Read our decision guide](#)

All applications Owned applications

Start typing a name or Application ID to filter these results

Display name	Application (client) ID	Created On	Certificates & secrets
Contoso Geo app			

2. Enter a **Name**, choose **Accounts in this organizational directory only** as the **Supported account type**. In **Redirect URIs**, specify **Public client / native (mobile & desktop)** then add <https://login.microsoftonline.com/common/oauth2/nativeclient> to the value. For more details please see [Azure AD Desktop app that calls web APIs: App registration](#). Then **Register** the application.

Register an application

* Name

The user-facing display name for this application (this can be changed later).

Contoso Geo app ✓

Supported account types

Who can use this application or access this API?

- Accounts in this organizational directory only (MS_AZURE_MAPS only - Single tenant)
 Accounts in any organizational directory (Any Azure AD directory - Multitenant)
 Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Public client/native (mobile ... ✓ https://login.microsoftonline.com/common/oauth2/nativeclient ✓

By proceeding, you agree to the Microsoft Platform Policies [↗](#)

Register

3. Navigate to **Authentication** and enable **Treat application as a public client**. This will enable device code authentication with Azure AD.

Advanced settings

Default client type (1)

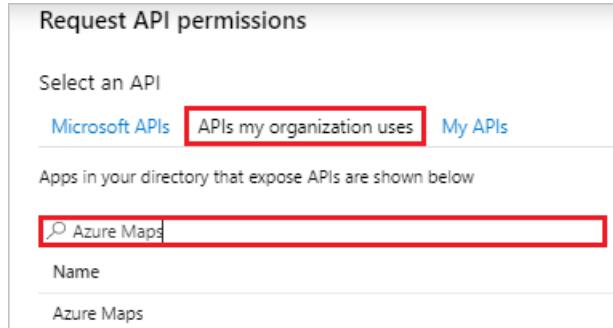
Treat application as a public client.

Required for the use of the following flows where a redirect URI is not used:



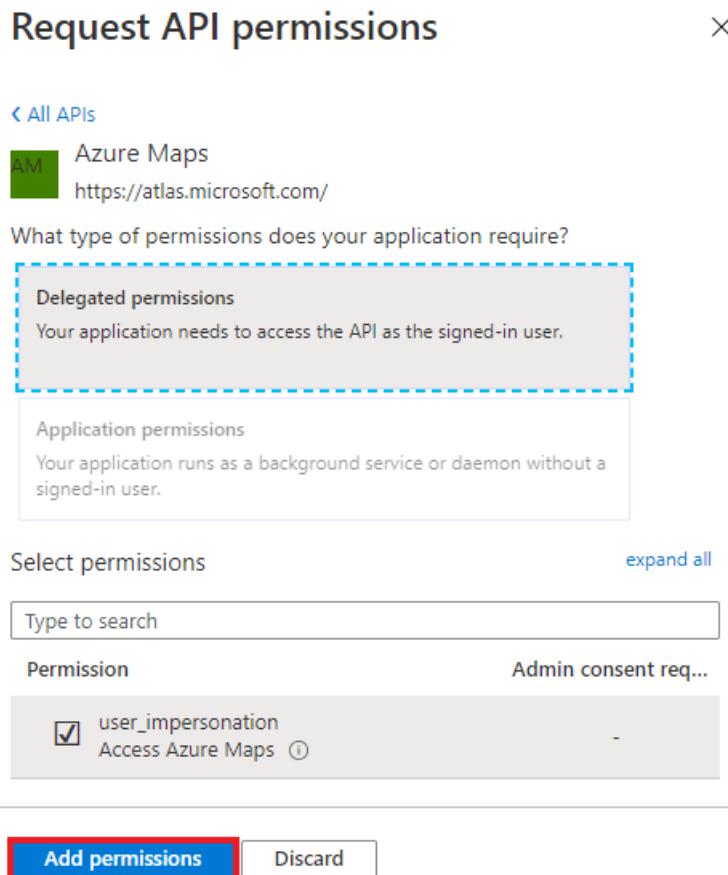
- Resource owner password credential (ROPC) [Learn more](#)
- Device code flow [Learn more](#)
- Integrated Windows Authentication (IWA) [Learn more](#)

4. To assign delegated API permissions to Azure Maps, go to the application. Then select **API permissions > Add a permission**. Under **APIs my organization uses**, search for and select **Azure Maps**.



The screenshot shows the 'Request API permissions' dialog. At the top, it says 'Select an API'. Below that are three tabs: 'Microsoft APIs' (disabled), 'APIs my organization uses' (selected, highlighted with a red border), and 'My APIs'. Under the 'APIs my organization uses' tab, it says 'Apps in your directory that expose APIs are shown below'. A list box contains 'Azure Maps', which is also highlighted with a red border. Below the list box, there are two rows: 'Name' and 'Azure Maps'.

5. Select the check box next to **Access Azure Maps**, and then select **Add permissions**.



The screenshot shows the 'Request API permissions' dialog for the 'Azure Maps' application. At the top, it says 'What type of permissions does your application require?'. There are two sections: 'Delegated permissions' (selected, highlighted with a blue dashed border) and 'Application permissions'. Under 'Delegated permissions', it says 'Your application needs to access the API as the signed-in user.' Below this, under 'Application permissions', it says 'Your application runs as a background service or daemon without a signed-in user.' At the bottom, there is a 'Select permissions' section with a 'Type to search' input field and a table. The table has columns 'Permission' and 'Admin consent req...'. One row is selected, showing 'user_impersonation' and 'Access Azure Maps (1)'. At the very bottom are two buttons: 'Add permissions' (highlighted with a red border) and 'Discard'.

6. Configure Azure role-based access control (Azure RBAC) for users or groups. See [Grant role-based access for users to Azure Maps](#).

7. Add code for acquiring token flow in the application, for implementation details see [Device code flow](#).

When acquiring tokens, reference the scope: `user_impersonation` which was selected on earlier steps.

TIP

Use Microsoft Authentication Library (MSAL) to acquire access tokens. See recommendations on [Desktop app that calls web APIs: Code configuration](#)

8. Compose the HTTP request with the acquired token from Azure AD, and sent request with a valid HTTP client.

Sample request

Here's a sample request body for uploading a simple Geofence represented as a circle geometry using a center point and a radius.

```
POST /mapData?api-version=2.0&dataFormat=geojson
Host: us.atlas.microsoft.com
x-ms-client-id: 30d7cc....9f55
Authorization: Bearer eyJ0e....HNIVN
```

The sample request body below is in GeoJSON:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [-122.126986, 47.639754]
      },
      "properties": {
        "geometryId": "001",
        "radius": 500
      }
    }
  ]
}
```

Sample response header

```
Operation-Location: https://us.atlas.microsoft.com/mapData/operations/{udid}?api-version=2.0
Access-Control-Expose-Headers: Operation-Location
```

Grant role-based access for users to Azure Maps

You grant *Azure role-based access control (Azure RBAC)* by assigning either an Azure AD group or security principals to one or more Azure Maps role definitions. To view Azure role definitions that are available for Azure Maps, go to [Access control \(IAM\)](#). Select **Roles**, and then search for roles that begin with *Azure Maps*.

- To efficiently manage a large amount of users' access to Azure Maps, see [Azure AD Groups](#).
- For users to be allowed to authenticate to the application, the users must be created in Azure AD. See [Add or Delete users using Azure AD](#).

Read more on [Azure AD](#) to effectively manage a directory for users.

1. Go to your [Azure Maps Account](#). Select **Access control (IAM) > Role assignment**.

The screenshot shows the Azure portal's Access control (IAM) blade for an Azure Maps account named "Contoso". The "Role assignments" tab is active, indicated by a red border around the tab name. Below the tabs, a message reads: "Manage access to Azure resources for users, groups, service principals and managed identities at this scope by creating role assignments. Learn more". A red box also highlights the "Number of role assignments for this subscription" link.

2. On the **Role assignments** tab, under **Role**, select a built in Azure Maps role definition such as **Azure Maps Data Reader** or **Azure Maps Data Contributor**. Under **Assign access to**, select **Azure AD user, group, or service principal**. Select the principal by name. Then select **Save**.
 - See details on [Assign Azure roles](#).

WARNING

Azure Maps built-in role definitions provide a very large authorization access to many Azure Maps REST APIs. To restrict APIs for users to a minimum, see [create a custom role definition and assign users](#) to the custom role definition. This will enable users to have the least privilege necessary for the application.

Next steps

Find the API usage metrics for your Azure Maps account:

[View usage metrics](#)

Manage the pricing tier of your Azure Maps account

5/12/2021 • 2 minutes to read • [Edit Online](#)

You can manage the pricing tier of your Azure Maps account through the Azure portal. You can also view or change your account's pricing tier after you create an [account](#).

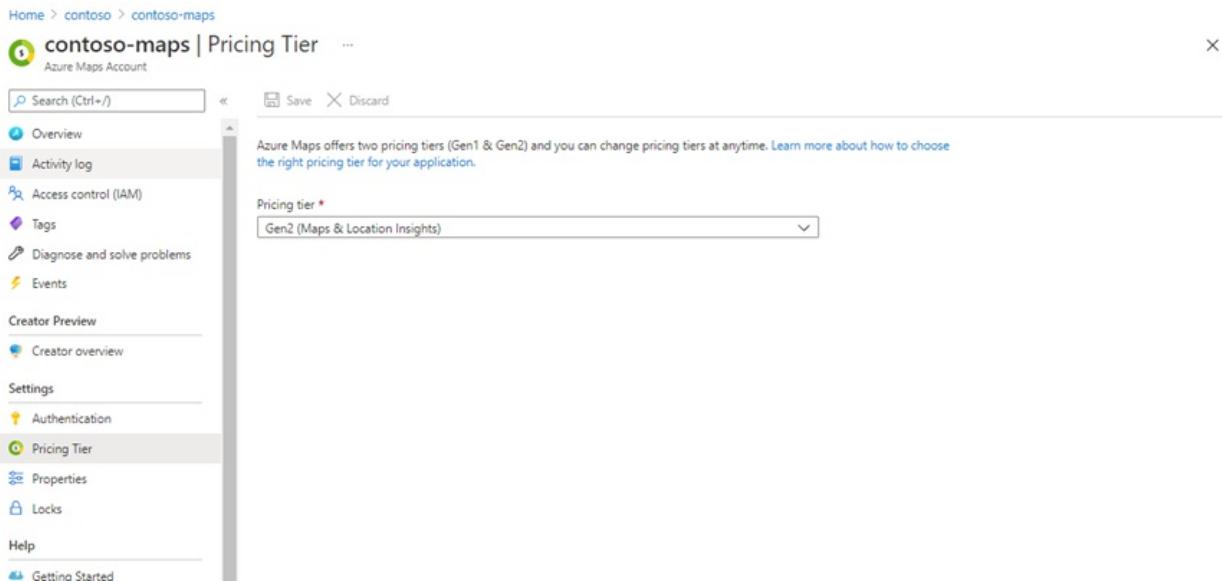
Get more information about [choosing the right pricing tier in Azure Maps](#).

NOTE

Switching to Gen 1 pricing tier is not available for Gen 2 Azure Maps Creator customers. Gen 1 Azure Maps Creator will be deprecated on 8/6/2021.

View your pricing tier

To view your chosen pricing tier, navigate to the **Pricing Tier** option in the settings menu.



Change a pricing tier

After you create your Azure Maps account, you can upgrade or downgrade the pricing tier for your Azure Maps account. To upgrade or downgrade, navigate to the **Pricing Tier** option in the settings menu. Select the pricing tier from drop down list. Note – current pricing tier will be default selection. Select the **Save** button to save your chosen pricing tier option.

NOTE

You don't have to generate new subscription keys or client ID (for Azure AD authentication) if you upgrade or downgrade the pricing tier for your Azure Maps account.

contoso-maps | Pricing Tier ...

Azure Maps Account

Search (Ctrl+ /) Save Discard

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Events

Creator Preview Creator overview

Settings Authentication Pricing Tier Properties Locks

Help Getting Started About

Azure Maps offers two pricing tiers (Gen1 & Gen2) and you can change pricing tiers at anytime. Learn more about how to choose the right pricing tier for your application.

Pricing tier *

Gen2 (Maps & Location Insights)

Recommended

Gen2 (Maps & Location Insights)

Other

Gen1 (S0)

Gen1 (S1)

Next steps

Learn how to see the API usage metrics for your Azure Maps account:

[View usage metrics](#)

Manage Azure Maps Creator

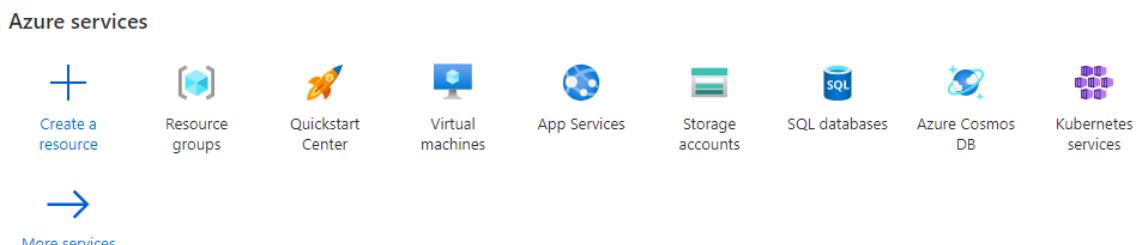
6/1/2021 • 2 minutes to read • [Edit Online](#)

You can use Azure Maps Creator to create private indoor map data. Using the Azure Maps API and the Indoor Maps module, you can develop interactive and dynamic indoor map web applications. For pricing information, see [Choose the right pricing tier in Azure Maps](#).

This article takes you through the steps to create and delete a Creator resource in an Azure Maps account.

Create Creator resource

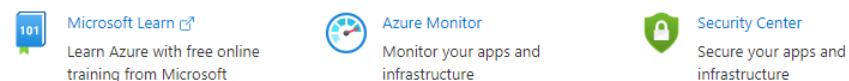
1. Sign in to the [Azure portal](#)
2. Navigate to the Azure portal menu. Select **All resources**, and then select your Azure Maps account.



Navigate



Tools



3. In the navigation pane, select **Creator overview**, and then select **Create**.

Home > Contoso

Contoso | Creator overview ...

Azure Maps Account

Search (Ctrl+ /) Create a Creator resource

Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Events

Creator

Creator overview

Settings

Authentication
Pricing Tier
Properties
Locks

Help

Getting Started
About

Bring your maps data

Creator makes it possible to create private maps and develop applications using Azure Maps API and SDK. To create and start using Creator, click Create. [Learn more](#)



Create

4. Enter the name, location, and map provisioning storage units for your Creator resource. Currently, Creator is supported only in the United States. Select **Review + create**.

Home > Azure Maps Accounts > Contoso >

Create a Creator resource ...

* Basics Tags Review + create

Creator makes it possible to create private maps and develop applications using Azure Maps API and SDK. [Learn more](#)

Project details

Subscription Subscription
Resource group resource-group
Azure Maps account name Contoso

Instance details

Creator name * contoso-indoor-maps ✓
Location * East US 2
Storage Units * 1 ✓
100 MiB

Review + create Next : Tags > Download a template for automation

5. Review your settings, and then select **Create**.

Create a Creator resource ...



* Basics Tags Review + create

Basics

Subscription	00000000-0000-0000-0000-000000000000
Resource group	resource-group
Azure Maps account name	Contoso
Creator name	contoso-indoor-maps
Location	East US 2
Storage Units	1

[Create](#)[< Previous : Tags](#)[Download a template for automation](#)

After the deployment completes, you'll see a page with a success or a failure message.



CreatorResource | Overview

[Overview](#)[Inputs](#)[Outputs](#)[Template](#)

We'd love your feedback! →

Your deployment is complete

Deployment name: CreatorResource
Subscription: Subscription
Resource group: resource-groupStart time: 5/21/2021, 12:18:35 PM
Correlation ID: 00000000-0000-0000-000000000000

Deployment details (Download)

Next steps

[Go to resource](#)

Security Center

Secure your apps and info
[Go to Azure security center](#)

Free Microsoft tutorials

[Start learning today >](#)

Work with an expert

Azure experts are service who can help manage yo and be your first line of s
[Find an Azure expert >](#)

6. Select **Go to resource**. Your Creator resource view page shows the status of your Creator resource and the chosen demographic region.

The screenshot shows the Azure portal interface for a resource named 'contoso-indoor-maps'. The top navigation bar includes 'Search (Ctrl+ /)', 'Edit', 'Delete', 'Refresh', and three dots for more options. On the far right is a close button ('X'). Below the search bar is a breadcrumb trail: 'Home > CreatorResource > contoso-indoor-maps (Contoso/contoso-indoor-maps) ...'. The main content area has a title 'contoso-indoor-maps (Contoso/contoso-indoor-maps)' followed by the subtitle 'Azure Maps Creator Preview Resource'. A left sidebar lists several sections: Overview (selected), Tags, Azure Maps Account (with 'Azure Maps Account' listed under it), Settings, Properties, Monitoring (with 'Alerts' and 'Metrics'), and Automation (with 'Tasks (preview)'). The 'Overview' section contains the following details:

Setting	Value
Location	East US 2
Subscription ID	00000000-0000-0000-0000-000000000000
Resource group	resource-group
Tags (change)	Click here to add tags

On the right side of the page, there are links for 'View Cost' and 'JSON View'. At the bottom right of the main content area is a small 'X' icon.

NOTE

To return to the Azure Maps account, select **Azure Maps Account** in the navigation pane.

Delete Creator resource

To delete the Creator resource:

1. In your Azure Maps account, select **Overview** under **Creator**.
2. Select **Delete**.

WARNING

When you delete the Creator resource of your Azure Maps account, you also delete the conversions, datasets, tilesets, and feature statesets that were created using Creator services.

 contoso-indoor-maps | Creator overview ...

Azure Maps Account

Search (Ctrl+/) < Edit **Delete** Refresh

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Events

Essentials

Location : East US 2 

Subscription ID : 00000000-0000-0000-0000-000000000000

Resource name : contoso-indoor-maps

Resource group : resource-group

Storage Units : 1

Tags (change) : Click here to add tags

Creator

Creator overview

Settings

- Authentication
- Pricing Tier
- Properties
- Locks

Help

- Getting Started
- About

3. You'll be asked to confirm deletion by typing in the name of your Creator resource. After the resource is deleted, you see a confirmation page that looks like the following:

Home > CreatorResource > contoso-indoor-maps (Contoso/contoso-indoor-maps) >

Are you sure you want to delete contoso-indoor-maps? ... 

 Warning! Deleting contoso-indoor-maps is irreversible. The action you're about to take can't be undone. Going further will delete it and all the items in it permanently.

TYPE THE AZURE MAPS CREATOR RESOURCE NAME

Delete **Cancel**

Authentication

Creator inherits Azure Maps Access Control (IAM) settings. All API calls for data access must be sent with authentication and authorization rules.

Creator usage data is incorporated in your Azure Maps usage charts and activity log. For more information, see [Manage authentication in Azure Maps](#).

IMPORTANT

We recommend using:

- Azure Active Directory (Azure AD) in all solutions that are built with an Azure Maps account using Creator services. For more information, on Azure AD, see [Azure AD authentication](#).
- Role-based access control settings (RBAC). Using these settings, map makers can act as the Azure Maps Data Contributor role, and Creator map data users can act as the Azure Maps Data Reader role. For more information, see [Authorization with role-based access control](#).

Access to Creator services

Creator services and services that use data hosted in Creator (for example, Render service), are accessible at a geographical URL. The geographical URL is determined by the location selected during creation. For example, if Creator is created in a region in the United States geographical location, all calls to the Conversion service must be submitted to `us.atlas.microsoft.com/conversions`. To view mappings of region to geographical location, see [Creator service geographic scope](#).

Also, all data imported into Creator should be uploaded into the same geographical location as the Creator resource. For example, if Creator is provisioned in the United States, all raw data should be uploaded via `us.atlas.microsoft.com/mapData/upload`.

Next steps

Introduction to Creator services for indoor mapping:

[Data upload](#)

[Data conversion](#)

[Dataset](#)

[Tileset](#)

[Feature State set](#)

Learn how to use the Creator services to render indoor maps in your application:

[Azure Maps Creator tutorial](#)

[Indoor map dynamic styling](#)

[Use the Indoor Maps module](#)

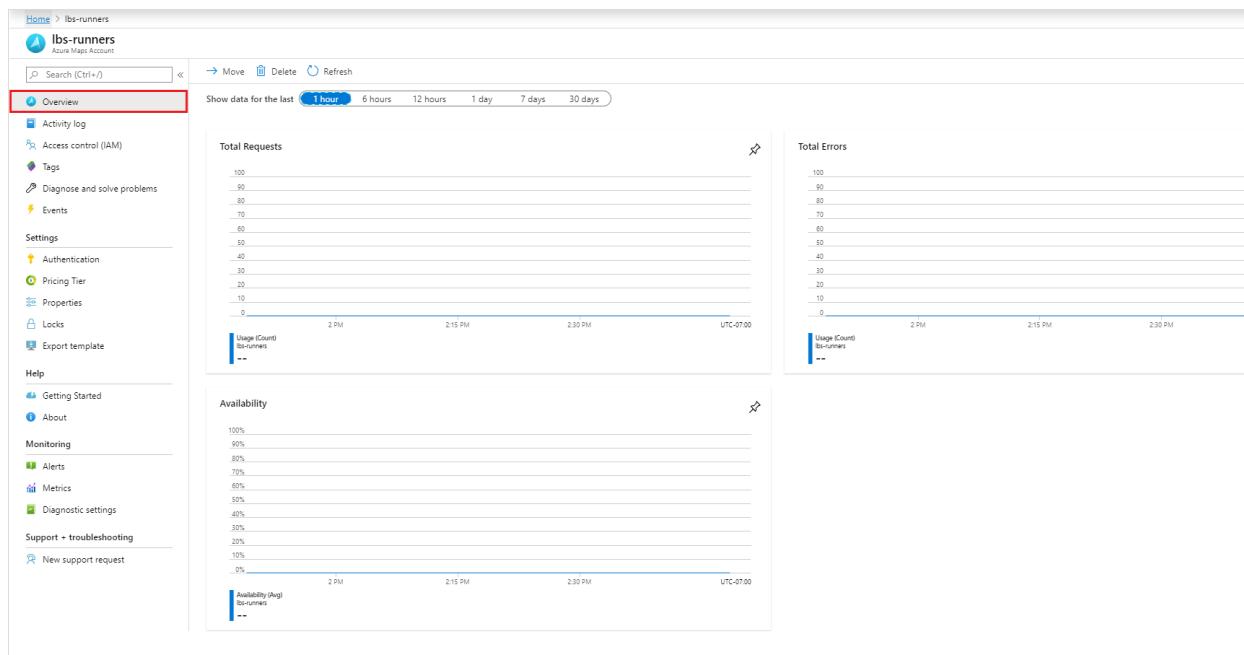
View Azure Maps API usage metrics

11/2/2020 • 2 minutes to read • [Edit Online](#)

This article shows you how to view the API usage metrics, for your Azure Maps account, in the [Azure portal](#). The metrics are shown in a convenient graph format along a customizable time duration.

View metric snapshot

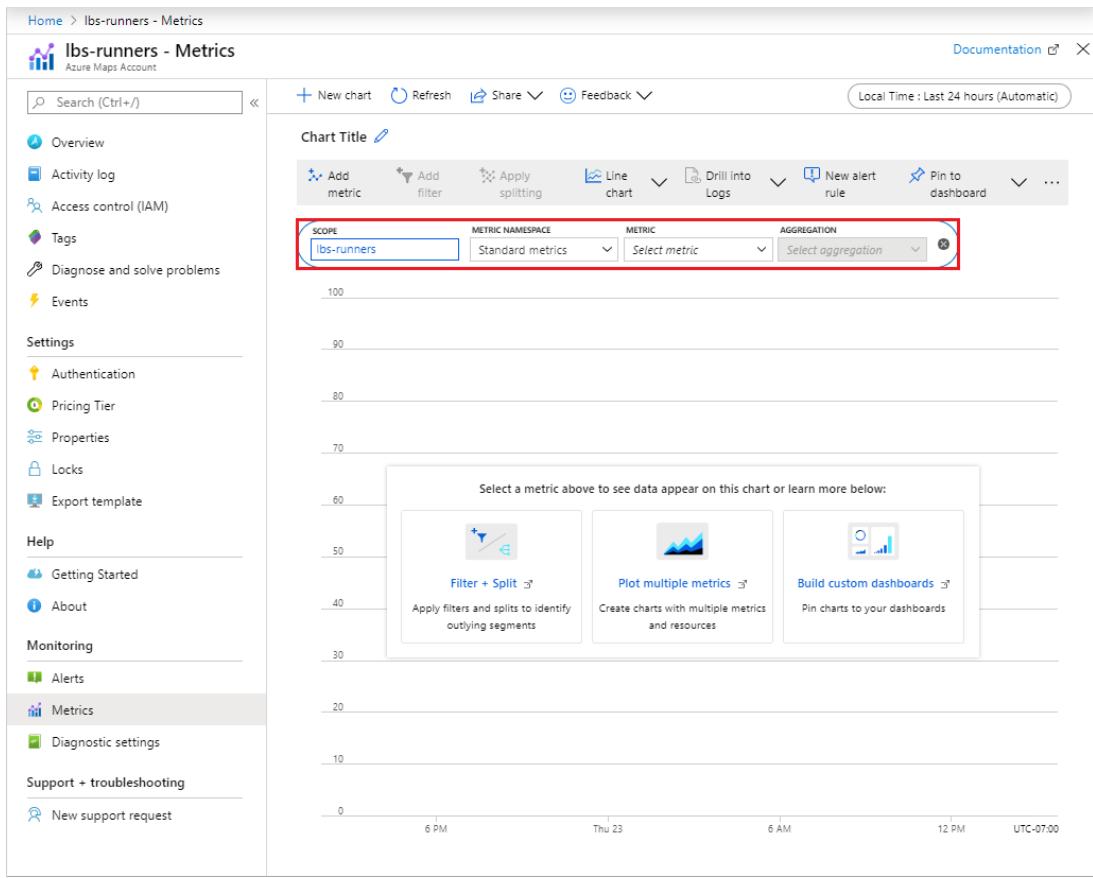
You can see some common metrics on the **Overview** page of your Maps account. It currently shows *Total Requests*, *Total Errors*, and *Availability* over a selectable time duration.



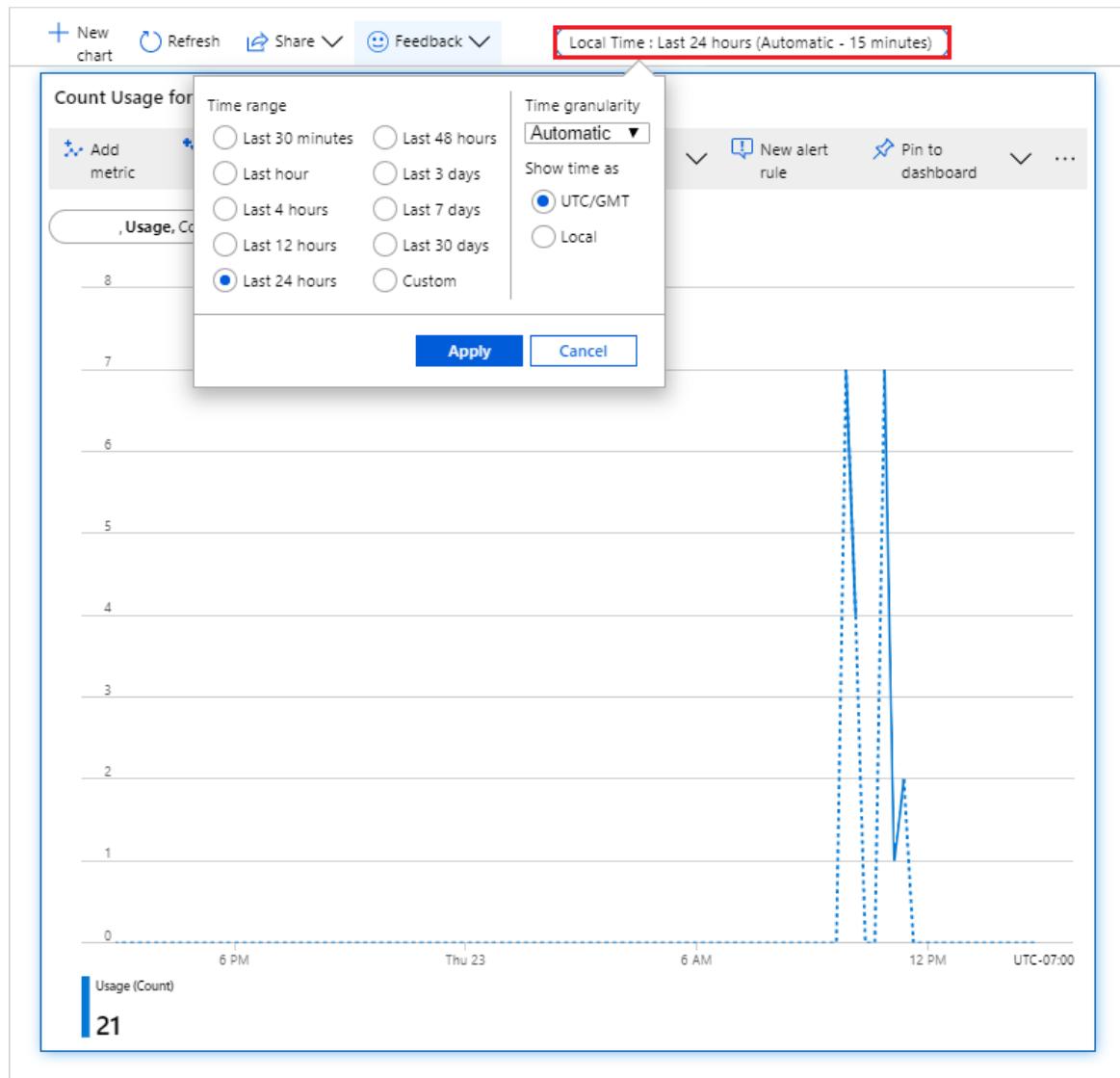
Continue to the next section if you need to customize these graphs for your particular analysis.

View detailed metrics

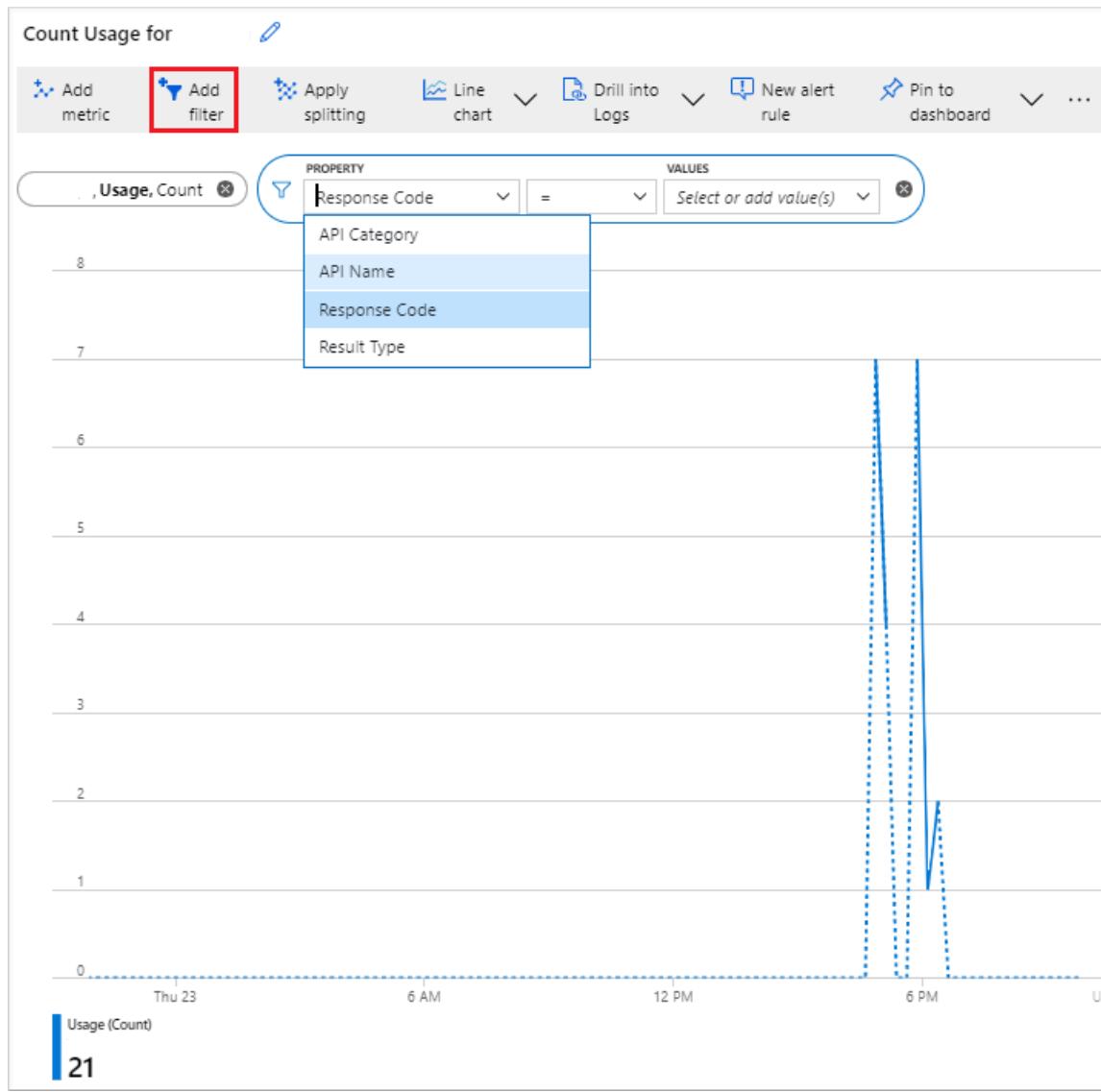
1. Sign in to your Azure subscription in the [portal](#).
2. Click the **All resources** menu item on the left-hand side and navigate to your *Azure Maps Account*.
3. Once your Maps account is open, click on the **Metrics** menu on the left.
4. On the **Metrics** pane, choose one of the following options:
 - a. **Availability** - which shows the *Average* of API availability over a period of time.
 - b. **Usage** - which shows how the usage *Count* for your account.



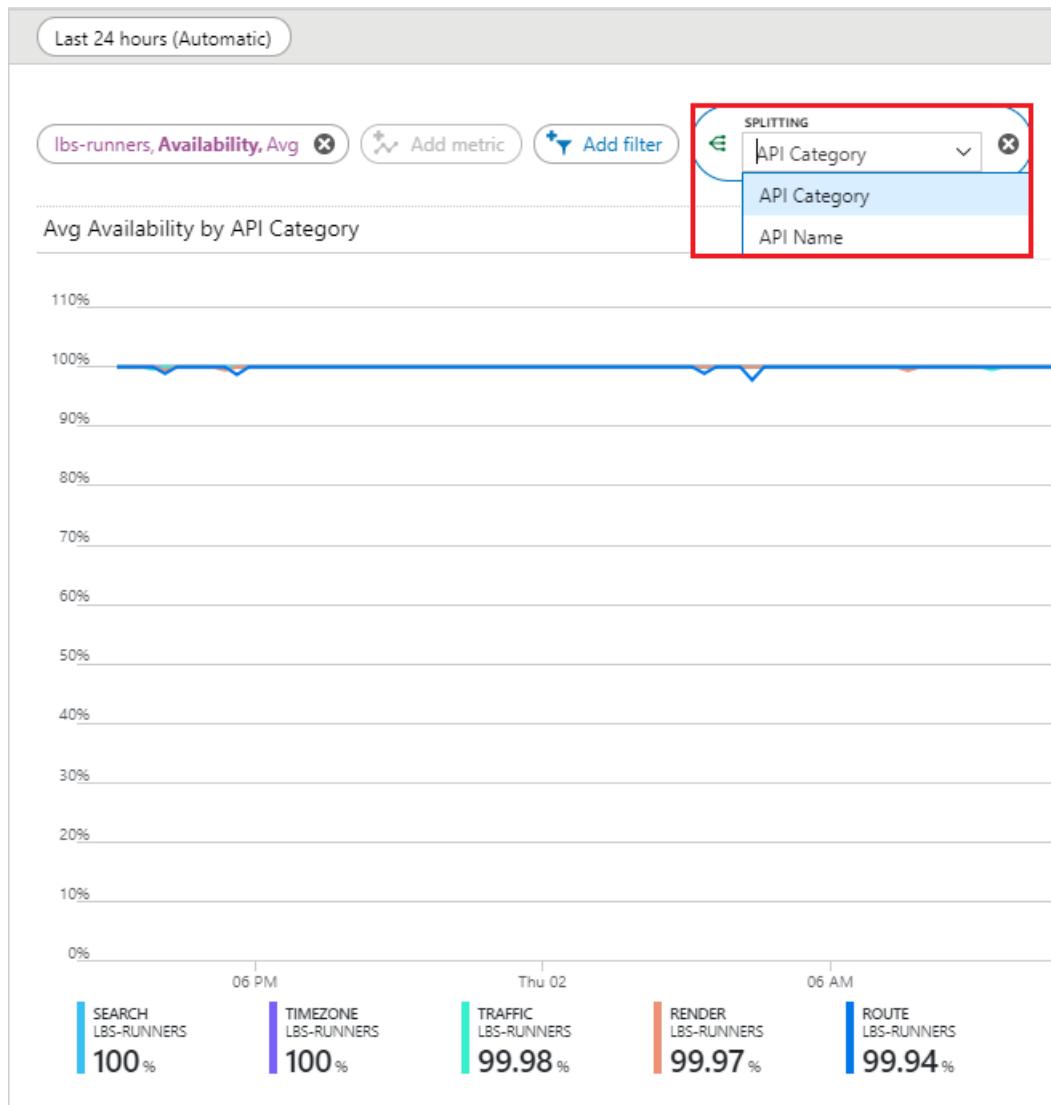
5. Next, you may select the *Time range* by clicking **Last 24 hours (Automatic)**. By default, the time range is set to 24 hours. After clicking, you'll see all selectable time ranges. You can select the *Time granularity* and choose to show the time as *local* or *GMT* in the same drop-down. Click **Apply**.



6. Once you add your metric, you can **Add filter** from the properties relevant to that metric. Then, select the value of the property that you want to see reflected on the graph.



7. You may also **Apply splitting** for your metric based on your selected metric property. It allows the graph to be split into multiple graphs, for each value of that property. In the following picture, the color of each graph corresponds to the property value shown at the bottom of the graph.



8. You may also observe multiple metrics on the same graph, simply by clicking on the **Add metric** button on top.

Next steps

Learn more about the Azure Maps APIs you want to track usage for:

[Azure Maps Web SDK How-To](#)

[Azure Maps Android SDK How-To](#)

[Azure Maps REST API documentation](#)

Search for a location using Azure Maps Search services

3/5/2021 • 8 minutes to read • [Edit Online](#)

The [Azure Maps Search Service](#) is a set of RESTful APIs designed to help developers search addresses, places, and business listings by name, category, and other geographic information. In addition to supporting traditional geocoding, services can also reverse geocode addresses and cross streets based on latitudes and longitudes. Latitude and longitude values returned by the search can be used as parameters in other Azure Maps services, such as [Route](#) and [Weather](#) services.

In this article, you'll learn how to:

- Request latitude and longitude coordinates for an address (geocode address location) by using the [Search Address API](#).
- Search for an address or Point of Interest (POI) using the [Fuzzy Search API](#).
- Make a [Reverse Address Search](#) to translate coordinate location to street address.
- Translate coordinate location into a human understandable cross street by using [Search Address Reverse Cross Street API](#). Most often, this is needed in tracking applications that receive a GPS feed from a device or asset, and wish to know where the coordinate is located.

Prerequisites

1. [Make an Azure Maps account](#)
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.

This tutorial uses the [Postman](#) application, but you may choose a different API development environment.

Request latitude and longitude for an address (geocoding)

In this example, we'll use the Azure Maps [Get Search Address API](#) to convert an address into latitude and longitude coordinates. This process is also called *geocoding*. In addition to returning the coordinates, the response will also return detailed address properties such as street, postal code, municipality, and country/region information.

TIP

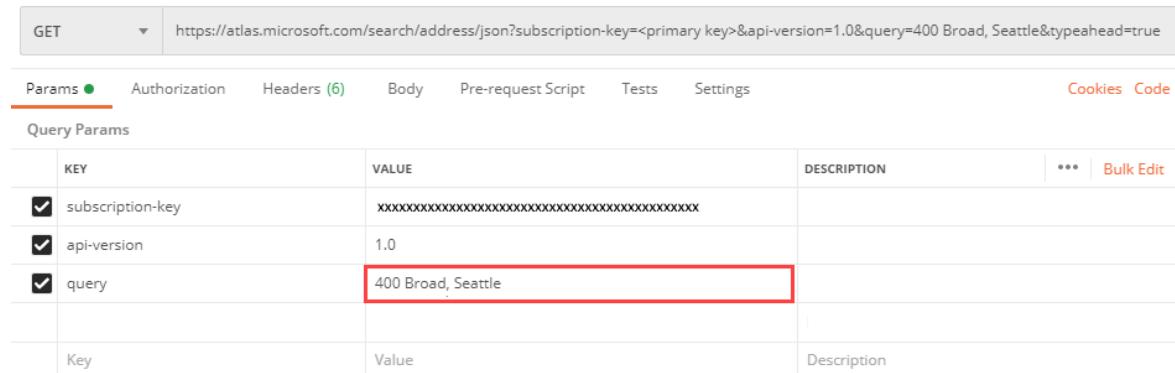
If you have a set of addresses to geocode, you can use the [Post Search Address Batch API](#) to send a batch of queries in a single API call.

1. Open the Postman app. Near the top of the Postman app, select **New**. In the **Create New** window, select **Collection**. Name the collection and select the **Create** button. You'll use this collection for the rest of the examples in this document.
2. To create the request, select **New** again. In the **Create New** window, select **Request**. Enter a **Request name** for the request. Select the collection you created in the previous step, and then select **Save**.
3. Select the **GET** HTTP method in the builder tab and enter the following URL. In this request, we're searching for a specific address: `400 Braod St, Seattle, WA 98109`. For this request, and other requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription

key.

```
https://atlas.microsoft.com/search/address/json?subscription-key={Azure-Maps-Primary-Subscription-key}&api-version=1.0&language=en-US&query=400 Broad St, Seattle, WA 98109
```

4. Click the blue **Send** button. The response body will contain data for a single location.
5. Now, we'll search an address that has more than one possible locations. In the **Params** section, change the `query` key to `400 Broad, Seattle`. Click the blue **Send** button.



The screenshot shows the Postman interface with a GET request to `https://atlas.microsoft.com/search/address/json?subscription-key=<primary key>&api-version=1.0&query=400 Broad, Seattle&typeahead=true`. The **Params** tab is active, displaying the following table:

KEY	VALUE	DESCRIPTION
subscription-key	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	
api-version	1.0	
query	400 Broad, Seattle	

6. Next, try setting the `query` key to `400 Broa`.
7. Click the **Send** button. You can now see that the response includes responses from multiple countries. To geobias results to the relevant area for your users, always add as many location details as possible to the request.

Using Fuzzy Search API

The Azure Maps [Fuzzy Search API](#) supports standard single line and free-form searches. We recommend that you use the Azure Maps Search Fuzzy API when you don't know your user input type for a search request. The query input can be a full or partial address. It can also be a Point of Interest (POI) token, like a name of POI, POI category or name of brand. Furthermore, to improve the relevance of your search results, the query results can be constrained by a coordinate location and radius, or by defining a bounding box.

TIP

Most Search queries default to `maxFuzzyLevel=1` to gain performance and reduce unusual results. You can adjust fuzziness levels by using the `maxFuzzyLevel` or `minFuzzyLevel` parameters. For more information on `maxFuzzyLevel` and a complete list of all optional parameters, see [Fuzzy Search URI Parameters](#)

Search for an address using Fuzzy Search

In this example, we'll use Fuzzy Search to search the entire world for `pizza`. Then, we'll show you how to search over the scope of a specific country. Finally, we'll show you how to use a coordinate location and radius to scope a search over a specific area, and limit the number of returned results.

IMPORTANT

To geobias results to the relevant area for your users, always add as many location details as possible. To learn more, see [Best Practices for Search](#).

1. Open the Postman app, click **New**, and select **Request**. Enter a **Request name** for the request. Select the collection you created in the previous section or created a new one, and then select **Save**.

2. Select the GET HTTP method in the builder tab and enter the following URL. For this request, and other requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key.

```
https://atlas.microsoft.com/search/fuzzy/json?api-version=1.0&subscription-key={Azure-Maps-Primary-Subscription-key}&language=en-US&query=pizza
```

NOTE

The `json` attribute in the URL path determines the response format. This article uses json for ease of use and readability. To find other supported response formats, see the `format` parameter definition in the [URI Parameter reference documentation](#).

3. Click **Send** and review the response body.

The ambiguous query string for "pizza" returned 10 **point of interest result** (POI) in both the "pizza" and "restaurant" categories. Each result includes details such as street address, latitude and longitude values, view port, and entry points for the location. The results are now varied for this query, and are not tied to any reference location.

In the next step, we'll use the `countrySet` parameter to specify only the countries/regions for which your application needs coverage. For a complete list of supported countries/regions, see [Search Coverage](#).

4. The default behavior is to search the entire world, potentially returning unnecessary results. Next, we'll search for pizza only the United States. Add the `countrySet` key to the **Params** section, and set its value to `us`. Setting the `countrySet` key to `us` will bound the results to the United States.

The screenshot shows the Postman interface for a GET request to the Microsoft Azure Maps search endpoint. The URL is `https://atlas.microsoft.com/search/fuzzy/json?api-version=1.0&subscription-key=<primary key>&query=pizza&countrySet=US`. The **Params** tab is selected, showing the following table:

KEY	VALUE	DESCRIPTION	...	Bi
<input checked="" type="checkbox"/> api-version	1.0			
<input checked="" type="checkbox"/> subscription-key	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx			
<input checked="" type="checkbox"/> query	pizza			
<input checked="" type="checkbox"/> countrySet	US			

A red box highlights the row for `countrySet` with the value `US`.

The results are now bounded by the country code and the query returns pizza restaurants in the United States.

5. To get an even more targeted search, you can search over the scope of a lat./lon. coordinate pair. In this example, we'll use the lat./lon. of the Seattle Space Needle. Since we only want to return results within a 400-meters radius, we'll add the `radius` parameter. Also, we'll add the `limit` parameter to limit the results to the five closest pizza places.

In the **Params** section, add the following key/value pairs:

KEY	VALUE
lat	47.620525
lon	-122.349274

KEY	VALUE
radius	400
limit	5

- Click **Send**. The response includes results for pizza restaurants near the Seattle Space Needle.

Search for a street address using Reverse Address Search

The Azure Maps [Get Search Address Reverse API](#) translates coordinates into human readable street addresses. This API is often used for applications that consume GPS feeds and want to discover addresses at specific coordinate points.

IMPORTANT

To geobias results to the relevant area for your users, always add as many location details as possible. To learn more, see [Best Practices for Search](#).

TIP

If you have a set of coordinate locations to reverse geocode, you can use [Post Search Address Reverse Batch API](#) to send a batch of queries in a single API call.

In this example, we'll be making reverse searches using a few of the optional parameters that are available. For the full list of optional parameters, see [Reverse Search Parameters](#).

- In the Postman app, click **New**, and select **Request**. Enter a **Request name** for the request. Select the collection you created in the first section or created a new one, and then select **Save**.
- Select the **GET** HTTP method in the builder tab and enter the following URL. For this request, and other requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key. The request should look like the following URL:

```
https://atlas.microsoft.com/search/address/reverse/json?api-version=1.0&subscription-key={Azure-Maps-Primary-Subscription-key}&language=en-US&query=47.591180,-122.332700&number=1
```

- Click **Send**, and review the response body. You should see one query result. The response includes key address information about Safeco Field.
- Now, we'll add the following key/value pairs to the **Params** section:

KEY	VALUE	RETURNS
number	1	The response may include the side of the street (Left/Right) and also an offset position for the number.
returnSpeedLimit	true	Returns the speed limit at the address.

KEY	VALUE	RETURNS
returnRoadUse	true	Returns road use types at the address. For all possible road use types, see Road Use Types .
returnMatchType	true	Returns the type of match. For all possible values, see Reverse Address Search Results

GET ▼ <https://atlas.microsoft.com/search/address/reverse/json?api-version=1.0&subscription-key=<primary key>&lat=47.591180&lon=-122.332700&number=1>

Params ● Auth Headers (6) Body Pre-req. Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> api-version	1.0			
<input checked="" type="checkbox"/> subscription-key	xx			
<input checked="" type="checkbox"/> language	en-US			
<input checked="" type="checkbox"/> query	47.591180,-122.332700			
<input checked="" type="checkbox"/> number	1			
<input checked="" type="checkbox"/> returnSpeedLimit	true			
<input checked="" type="checkbox"/> returnRoadUse	true			
<input checked="" type="checkbox"/> returnMatchType	true			

Key Value Description

5. Click **Send**, and review the response body.

6. Next, we'll add the `entityType` key, and set its value to `Municipality`. The `entityType` key will override the `returnMatchType` key in the previous step. We'll also need to remove `returnSpeedLimit` and `returnRoadUse` since we're requesting information about the municipality. For all possible entity types, see [Entity Types](#).

GET ▼ <https://atlas.microsoft.com/search/address/reverse/json?api-version=1.0&subscription-key=<primary key>&lat=47.591180&lon=-122.332700&number=1&entityType=Municipality>

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> api-version	1.0
<input checked="" type="checkbox"/> subscription-key	xx
<input checked="" type="checkbox"/> language	en-US
<input checked="" type="checkbox"/> query	47.591180,-122.332700
<input checked="" type="checkbox"/> number	1
<input type="checkbox"/> returnSpeedLimit	true
<input type="checkbox"/> returnRoadUse	true
<input type="checkbox"/> returnMatchType	true
<input checked="" type="checkbox"/> entityType	Municipality

Key Value

7. Click **Send**. Compare the results to the results returned in step 5. Because the requested entity type is now `municipality`, the response does not include street address information. Also, the returned `geometryId` can be used to request boundary polygon through Azure Maps Get [Search Polygon API](#).

TIP

To get more information on these parameters, as well as to learn about others, see the [Reverse Search Parameters section](#).

Search for cross street using Reverse Address Cross Street Search

In this example, we'll search for a cross street based on the coordinates of an address.

1. In the Postman app, click **New**, and select **Request**. Enter a **Request name** for the request. Select the collection you created in the first section or created a new one, and then select **Save**.
2. Select the **GET** HTTP method in the builder tab and enter the following URL. For this request, and other requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key. The request should look like the following URL:

```
https://atlas.microsoft.com/search/address/reverse/crossstreet/json?api-version=1.0&subscription-key={Azure-Maps-Primary-Subscription-key}&language=en-US&query=47.591180,-122.332700
```

The screenshot shows the Postman interface with a GET request. The URL is set to `https://atlas.microsoft.com/search/address/reverse/crossstreet/json?subscription-key=xxxxxxxxxxxxxx`. The 'Params' tab is selected, showing the following query parameters:

KEY	VALUE	DESCRIPTION
subscription-key	xxxxxxxxxxxxxxxxxxxxxxxxxxxx	
api-version	1.0	
query	47.591180,-122.332700	
language	en-US	
Key	Value	Description

3. Click **Send**, and review the response body. You'll notice that the response contains a `crossStreet` value of `South Atlantic Street`.

Next steps

[Azure Maps Search Service REST API](#)

[Azure Maps Search Service Best Practices](#)

Best practices for Azure Maps Search Service

11/2/2020 • 14 minutes to read • [Edit Online](#)

Azure Maps [Search Service](#) includes APIs that offer various capabilities to help developers to search addresses, places, business listings by name or category, and other geographic information. For example, [Fuzzy Search API](#) allows users to search for an address or Point of Interest (POI).

This article explains how to apply sound practices when you call data from Azure Maps Search Service. You'll learn how to:

- Build queries to return relevant matches
- Limit search results
- Learn the differences between result types
- Read the address search-response structure

Prerequisites

1. [Make an Azure Maps account](#)
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.

This article uses the [Postman app](#) to build REST calls, but you can choose any API development environment.

Best practices to geocode addresses

When you search for a full or partial address by using Azure Maps Search Service, the API reads keywords from your search query. Then it returns the longitude and latitude coordinates of the address. This process is called *geocoding*.

The ability to geocode in a country/region depends on the availability of road data and the precision of the geocoding service. For more information about Azure Maps geocoding capabilities by country or region, see [Geocoding coverage](#).

Limit search results

Azure Maps Search API can help you limit search results appropriately. You limit results so that you can display relevant data to your users.

NOTE

The search APIs support more parameters than just the ones that this article discusses.

Geobiased search results

To geobias results to the relevant area for your user, always add as many location details as possible. You might want to restrict the search results by specifying some input types:

- Set the `countrySet` parameter. You can set it to `US,FR`, for example. By default, the API searches the entire world, so it can return unnecessary results. If your query has no `countrySet` parameter, then the search might return inaccurate results. For example, a search for a city named *Bellevue* returns results from the USA and France because both countries/regions contain a city named *Bellevue*.
- You can use the `btmRight` and `topleft` parameters to set the bounding box. These parameters restrict the search to a specific area on the map.

- To influence the area of relevance for the results, define the `lat` and `lon` coordinate parameters. Use the `radius` parameter to set the radius of the search area.

Fuzzy search parameters

We recommend that you use the Azure Maps [Search Fuzzy API](#) when you don't know your user inputs for a search query. For example, input from the user could be an address or the type of Point of Interest (POI), like *shopping mall*. The API combines POI searching and geocoding into a canonical *single-line search*:

- The `minFuzzyLevel` and `maxFuzzyLevel` parameters help return relevant matches even when query parameters don't exactly match the information that the user wants. To maximize performance and reduce unusual results, set search queries to defaults of `minFuzzyLevel=1` and `maxFuzzyLevel=2`.

For example, when the `maxFuzzyLevel` parameter is set to 2, the search term *restrant* is matched to *restaurant*. You can override the default fuzzy levels when you need to.

- Use the `idxSet` parameter to prioritize the exact set of result types. To prioritize an exact set of results, you can submit a comma-separated list of indexes. In your list, the item order doesn't matter. Azure Maps supports the following indexes:
 - `Addr` - **Address ranges**: Address points that are interpolated from the beginning and end of the street. These points are represented as address ranges.
 - `Geo` - **Geographies**: Administrative divisions of land. A geography can be a country/region, state, or city, for example.
 - `PAD` - **Point addresses**: Addresses that include a street name and number. Point addresses can be found in an index. An example is *Soquel Dr 2501*. A point address provides the highest level of accuracy available for addresses.
 - `POI` - **Points of interest**: Points on a map that are considered to be worth attention or that might be interesting. The [Search Address API](#) doesn't return POIs.
 - `Str` - **Streets**: Streets on the map.
 - `xstr` - **Cross streets or intersections**: Junctions or places where two streets intersect.

Usage examples

- `idxSet=POI` - Search POIs only.
- `idxSet=PAD,Addr` - Search addresses only. `PAD` indicates the point address, and `Addr` indicates the address range.

Reverse-geocode and filter for a geography entity type

When you do a reverse-geocode search in the [Search Address Reverse API](#), the service can return polygons for administrative areas. For example, you might want to fetch the area polygon for a city. To narrow the search to specific geography entity types, include the `entityType` parameter in your requests.

The resulting response contains the geography ID and the entity type that was matched. If you provide more than one entity, then the endpoint returns the *smallest entity available*. You can use the returned geometry ID to get the geography's geometry through the [Search Polygon service](#).

Sample request

```
https://atlas.microsoft.com/search/address/reverse/json?api-version=1.0&subscription-key={subscription-key}&query=47.6394532,-122.1304551&language=en-US&entityType=Municipality
```

Response

```
{
  "summary": {
    "queryTime": 14,
    "numResults": 1
  },
  "addresses": [
    {
      "address": {
        "routeNumbers": [],
        "countryCode": "US",
        "countrySubdivision": "WA",
        "countrySecondarySubdivision": "King",
        "countryTertiarySubdivision": "Seattle East",
        "municipality": "Redmond",
        "country": "United States",
        "countryCodeISO3": "USA",
        "countrySubdivisionName": "Washington"
      },
      "position": "47.639454,-122.130455",
      "dataSources": {
        "geometry": {
          "id": "00005557-4100-3c00-0000-0000596ae571"
        }
      },
      "entityType": "Municipality"
    }
  ]
}
```

Set the results language

Use the `language` parameter to set the language for the returned search results. If the request doesn't set the language, then by default Search Service uses the most common language in the country or region. When no data is available in the specified language, the default language is used.

For more information, see [Azure Maps supported languages](#).

Use predictive mode (automatic suggestions)

To find more matches for partial queries, set the `typeahead` parameter to `true`. This query is interpreted as a partial input, and the search enters predictive mode. If you don't set the `typeahead` parameter to `true`, then the service assumes that all relevant information has been passed in.

In the following sample query, the Search Address service is queried for *Microso*. Here, the `typeahead` parameter set to `true`. The response shows that the search service interpreted the query as partial query. The response contains results for an automatically suggested query.

Sample query

```
https://atlas.microsoft.com/search/address/json?subscription-key={subscription-key}&api-version=1.0&typeahead=true&countrySet=US&lat=47.6370891183&lon=-122.123736172&query=Microsoft
```

Response

```
{
  "summary": {
    "query": "microsoft",
    "queryType": "NON_NEAR",
    "queryTime": 18,
    "numResults": 7,
    "offset": 0,
    "totalResults": 7,
    "fuzzyLevel": 1,
    "geoBias": {
```

```
        "lat": 47.6370891183,
        "lon": -122.123736172
    },
},
"results": [
{
    "type": "Street",
    "id": "US/STR/p0/9438784",
    "score": 2.594099998474121,
    "dist": 314.0590106663596,
    "address": {
        "streetName": "Microsoft Way",
        "municipalitySubdivision": "Redmond",
        "municipality": "Redmond",
    },
    "position": {
        "lat": 47.63988,
        "lon": -122.12438
    },
    "viewport": {
        "topLeftPoint": {
            "lat": 47.64223,
            "lon": -122.1256,
            "valid": true
        },
        "btmRightPoint": {
            "lat": 47.63748,
            "lon": -122.12309,
            "valid": true
        }
    }
},
{
    "type": "Street",
    "id": "US/STR/p0/1756074",
    "score": 2.592679977416992,
    "dist": 876.0272035824189,
    "address": {
        "streetName": "Microsoft Road",
        "municipalitySubdivision": "Redmond",
        "municipality": "Redmond",
        "countrySecondarySubdivision": "King",
        "countryTertiarySubdivision": "Seattle East",
        "countrySubdivision": "WA",
        "countrySubdivisionName": "Washington",
        "postalCode": "98052",
        "countryCode": "US",
        "country": "United States",
        "countryCodeISO3": "USA",
        "freeformAddress": "Microsoft Road, Redmond, WA 98052"
    },
    "position": {
        "lat": 47.64032,
        "lon": -122.1344
    },
    "viewport": {
        "topLeftPoint": {
            "lat": 47.64253,
            "lon": -122.13535,
            "valid": true
        },
        "btmRightPoint": {
            "lat": 47.63816,
            "lon": -122.13305,
            "valid": true
        }
    }
},
{
    "type": "Street"
}
]
```

```
        "type": "Street",
        "id": "US/STR/p0/1470668",
        "score": 2.5290400981903076,
        "dist": 2735.4883918101486,
        "address": {
            "streetName": "Microsoft West Campus Road",
            "municipalitySubdivision": "Redmond",
            "municipality": "Bellevue",
            "countrySecondarySubdivision": "King",
            "countryTertiarySubdivision": "Seattle East",
            "countrySubdivision": "WA",
            "countrySubdivisionName": "Washington",
            "postalCode": "98007",
            "countryCode": "US",
            "country": "United States",
            "countryCodeISO3": "USA",
            "freeformAddress": "Microsoft West Campus Road, Bellevue, WA 98007"
        },
        "position": {
            "lat": 47.65784,
            "lon": -122.14335
        },
        "viewport": {
            "topLeftPoint": {
                "lat": 47.65785,
                "lon": -122.14335,
                "valid": true
            },
            "btmRightPoint": {
                "lat": 47.65784,
                "lon": -122.14325,
                "valid": true
            }
        }
    },
    {
        "type": "Street",
        "id": "US/STR/p0/12812615",
        "score": 2.527509927749634,
        "dist": 2870.9579016916873,
        "address": {
            "streetName": "Microsoft West Campus Road",
            "municipalitySubdivision": "Redmond",
            "municipality": "Redmond",
            "countrySecondarySubdivision": "King",
            "countryTertiarySubdivision": "Seattle East",
            "countrySubdivision": "WA",
            "countrySubdivisionName": "Washington",
            "postalCode": "98052",
            "countryCode": "US",
            "country": "United States",
            "countryCodeISO3": "USA",
            "freeformAddress": "Microsoft West Campus Road, Redmond, WA 98052"
        },
        "position": {
            "lat": 47.66034,
            "lon": -122.1404
        },
        "viewport": {
            "topLeftPoint": {
                "lat": 47.66039,
                "lon": -122.14325,
                "valid": true
            },
            "btmRightPoint": {
                "lat": 47.65778,
                "lon": -122.13749,
                "valid": true
            }
        }
    }
]
```

```
        }
    },
{
    "type": "Street",
    "id": "US/STR/p0/197588",
    "score": 2.4630401134490967,
    "dist": 878.1404663812472,
    "address": {
        "streetName": "157th Avenue Northeast",
        "municipalitySubdivision": "Redmond",
        "municipality": "Redmond",
        "countrySecondarySubdivision": "King",
        "countryTertiarySubdivision": "Seattle East",
        "countrySubdivision": "WA",
        "countrySubdivisionName": "Washington",
        "postalCode": "98052",
        "extendedPostalCode": "980525344, 980525398, 980525399",
        "countryCode": "US",
        "country": "United States",
        "countryCodeISO3": "USA",
        "freeformAddress": "157th Avenue Northeast, Redmond, WA 98052"
    },
    "position": {
        "lat": 47.64351,
        "lon": -122.13056
    },
    "viewport": {
        "topLeftPoint": {
            "lat": 47.64473,
            "lon": -122.13058,
            "valid": true
        },
        "btmRightPoint": {
            "lat": 47.6425,
            "lon": -122.13016,
            "valid": true
        }
    }
},
{
    "type": "Street",
    "id": "US/STR/p0/3033991",
    "score": 2.0754499435424805,
    "dist": 3655467.8844475765,
    "address": {
        "streetName": "Microsoft Way",
        "municipalitySubdivision": "Yorkmount, Charlotte",
    },
    "position": {
        "lat": 35.14267,
        "lon": -80.91824
    },
    "viewport": {
        "topLeftPoint": {
            "lat": 35.14287,
            "lon": -80.91839,
            "valid": true
        },
        "btmRightPoint": {
            "lat": 35.14267,
            "lon": -80.91814,
            "valid": true
        }
    }
},
{
    "type": "Street",
    "id": "US/STR/p0/8395877",
    "score": 2.0754499435424805,
```

```
        "score": 2.0754499435424805,
        "dist": 3655437.0037482483,
        "address": {
            "streetName": "Microsoft Way",
            "municipalitySubdivision": "Charlotte",
            "municipality": "Charlotte",
            "countrySecondarySubdivision": "Mecklenburg",
            "countryTertiarySubdivision": "Township 1 Charlotte",
            "countrySubdivision": "NC",
            "countrySubdivisionName": "North Carolina",
            "postalCode": "28273",
            "extendedPostalCode": "282738105, 282738106, 282738108, 2827382, 282738200",
            "countryCode": "US",
            "country": "United States",
            "countryCodeISO3": "USA",
            "freeformAddress": "Microsoft Way, Charlotte, NC 28273"
        },
        "position": {
            "lat": 35.14134,
            "lon": -80.9198
        },
        "viewport": {
            "topLeftPoint": {
                "lat": 35.14274,
                "lon": -80.92159,
                "valid": true
            },
            "btmRightPoint": {
                "lat": 35.14002,
                "lon": -80.91824,
                "valid": true
            }
        }
    }
]
```

Encode a URI to handle special characters

To find cross street addresses, you must encode the URI to handle special characters in the address. Consider this address example: *1st Avenue & Union Street, Seattle*. Here, encode the ampersand character (&) before you send the request.

We recommend that you encode character data in a URI. In a URI, you encode all characters by using a percentage sign (%) and a two-character hexadecimal value that corresponds to the characters' UTF-8 code.

Usage examples

Start with this address:

```
query=1st Avenue & E 111th St, New York
```

Encode the address:

```
query=1st%20Avenue%20%26%20E%20111th%20St%2C%20New%20York
```

You can use the following methods.

JavaScript or TypeScript:

```
encodeURIComponent(query)
```

C# or Visual Basic:

```
Uri.EscapeDataString(query)
```

Java:

```
URLEncoder.encode(query, "UTF-8")
```

Python:

```
import urllib.parse  
urllib.parse.quote(query)
```

C++:

```
#include <curl/curl.h>  
curl_easy_escape(query)
```

PHP:

```
urlencode(query)
```

Ruby:

```
CGI::escape(query)
```

Swift:

```
query.stringByAddingPercentEncodingWithAllowedCharacters(.URLHostAllowedCharacterSet())
```

Go:

```
import ("net/url")  
url.QueryEscape(query)
```

Best practices for POI searching

In a POI search, you can request POI results by name. For example, you can search for a business by name.

We strongly recommend that you use the `countrySet` parameter to specify countries/regions where your application needs coverage. The default behavior is to search the entire world. This broad search might return unnecessary results, and the search might take a long time.

Brand search

To improve the relevance of the results and the information in the response, a POI search response includes brand information. You can use this information to further parse the response.

In a request, you can submit a comma-separated list of brand names. Use the list to restrict the results to specific brands by setting the `brandset` parameter. In your list, item order doesn't matter. When you provide multiple brand lists, the results that are returned must belong to at least one of your lists.

To explore brand searching, let's make a [POI category search](#) request. In the following example, we look for gas stations near the Microsoft campus in Redmond, Washington. The response shows brand information for each POI that was returned.

Sample query

```
https://atlas.microsoft.com/search/poi/json?subscription-key={subscription-key}&api-version=1.0&query=gas%20station&limit=3&lat=47.6413362&lon=-122.1327968
```

Response

```
{  
    "summary": {  
        "query": "gas station",  
        "queryType": "NON_NEAR",  
        "queryTime": 276,  
        "numResults": 3,  
        "offset": 0,  
        "totalResults": 762680,  
        "fuzzyLevel": 1,  
        "geoBias": {  
            "lat": 47.6413362,  
            "lon": -122.1327968  
        }  
    },  
    "results": [  
        {  
            "type": "POI",  
            "id": "US/POI/p0/8831765",  
            "score": 5.6631999015808105,  
            "dist": 1037.0280221303253,  
            "info": "search:ta:840531000004190-US",  
            "poi": {  
                "name": "Chevron",  
                "phone": "+(1)-(425)-6532200",  
                "brands": [  
                    {  
                        "name": "Chevron"  
                    }  
                ],  
                "categorySet": [  
                    {  
                        "id": 7311  
                    }  
                ],  
                "url": "www.chevron.com",  
                "categories": [  
                    "petrol station"  
                ],  
                "classifications": [  
                    {  
                        "code": "PETROL_STATION",  
                        "names": [  
                            {  
                                "nameLocale": "en-US",  
                                "name": "petrol station"  
                            }  
                        ]  
                    }  
                ]  
            },  
            "address": {  
                "streetNumber": "2444",  
                "streetName": "Bel Red Rd",  
                "municipalitySubdivision": "Northeast Bellevue, Bellevue",  
            },  
            "geocodes": [  
                {  
                    "lat": 47.6413362,  
                    "lon": -122.1327968  
                }  
            ]  
        }  
    ]  
}
```

```
        "position": {
            "lat": 47.63201,
            "lon": -122.13281
        },
        "viewport": {
            "topLeftPoint": {
                "lat": 47.63291,
                "lon": -122.13414,
                "valid": true
            },
            "bottomRightPoint": {
                "lat": 47.63111,
                "lon": -122.13148,
                "valid": true
            }
        },
        "entryPoints": [
            {
                "type": "main",
                "position": {
                    "lat": 47.63222,
                    "lon": -122.13312,
                    "valid": true
                }
            }
        ]
    },
    {
        "type": "POI",
        "id": "US/POI/p0/8831752",
        "score": 5.662710189819336,
        "dist": 1330.1278248163273,
        "info": "search:ta:840539001100326-US",
        "poi": {
            "name": "76",
            "phone": "+(1)-(425)-7472126",
            "brands": [
                {
                    "name": "76"
                }
            ],
            "categorySet": [
                {
                    "id": 7311
                }
            ],
            "url": "www.76.com",
            "categories": [
                "petrol station"
            ],
            "classifications": [
                {
                    "code": "PETROL_STATION",
                    "names": [
                        {
                            "nameLocale": "en-US",
                            "name": "petrol station"
                        }
                    ]
                }
            ]
        },
        "address": {
            "streetNumber": "2421",
            "streetName": "148Th Ave Ne",
            "municipalitySubdivision": "Redmond, Bridle Trails, Bellevue",
            "municipality": "Redmond, Bellevue",
            "countrySecondarySubdivision": "King",
            "countryTertiarySubdivision": "Seattle East",
            "countryPrimarySubdivision": "Washington"
        }
    }
]
```

```
        "countrySubdivision": "WA",
        "countrySubdivisionName": "Washington",
        "postalCode": "98007",
        "countryCode": "US",
        "country": "United States",
        "countryCodeISO3": "USA",
        "freeformAddress": "2421 148Th Ave Ne, Bellevue, WA 98007",
        "localName": "Bellevue"
    },
    "position": {
        "lat": 47.63187,
        "lon": -122.14365
    },
    "viewport": {
        "topLeftPoint": {
            "lat": 47.63277,
            "lon": -122.14498,
            "valid": true
        },
        "bottomRightPoint": {
            "lat": 47.63097,
            "lon": -122.14232,
            "valid": true
        }
    },
    "entryPoints": [
        {
            "type": "minor",
            "position": {
                "lat": 47.63187,
                "lon": -122.14374,
                "valid": true
            }
        },
        {
            "type": "main",
            "position": {
                "lat": 47.63186,
                "lon": -122.14313,
                "valid": true
            }
        }
    ]
},
{
    "type": "POI",
    "id": "US/POI/p0/8831764",
    "score": 5.662449836730957,
    "dist": 1458.645407416307,
    "info": "search:ta:840539000488527-US",
    "poi": {
        "name": "BROWN BEAR CAR WASH",
        "phone": "+(1)-(425)-6442868",
        "brands": [
            {
                "name": "Texaco"
            }
        ],
        "categorySet": [
            {
                "id": 7311
            }
        ],
        "url": "www.texaco.com/",
        "categories": [
            "petrol station"
        ],
        "classifications": [
            {

```

```

        "code": "PETROL_STATION",
        "names": [
            {
                "nameLocale": "en-US",
                "name": "petrol station"
            }
        ]
    },
    "address": {
        "streetNumber": "15248",
        "streetName": "Bel Red Rd",
        "municipalitySubdivision": "Redmond",
    },
    "position": {
        "lat": 47.62843,
        "lon": -122.13628
    },
    "viewport": {
        "topLeftPoint": {
            "lat": 47.62933,
            "lon": -122.13761,
            "valid": true
        },
        "bottomRightPoint": {
            "lat": 47.62753,
            "lon": -122.13495,
            "valid": true
        }
    },
    "entryPoints": [
        {
            "type": "main",
            "position": {
                "lat": 47.62827,
                "lon": -122.13628,
                "valid": true
            }
        }
    ]
}
]
}

```

Airport search

By using the Search POI API, you can look for airports by using their official code. For example, you can use *SEA* to find the Seattle-Tacoma International Airport:

```
https://atlas.microsoft.com/search/poi/json?subscription-key={subscription-key}&api-version=1.0&query=SEA
```

Nearby search

To retrieve POI results around a specific location, you can try using the [Search Nearby API](#). The endpoint returns only POI results. It doesn't take in a search query parameter.

To limit the results, we recommend that you set the radius.

Understanding the responses

Let's find an address in Seattle by making an address-search request to the Azure Maps Search Service. In the following request URL, we set the `countrySet` parameter to `us` to search for the address in the USA.

Sample query

```
https://atlas.microsoft.com/search/address/json?subscription-key={subscription-key}&api-version=1&query=400%20Broad%20Street%2C%20Seattle%2C%20WA&countrySet=US
```

Supported types of results

- **Point Address:** Points on a map that have a specific address with a street name and number. Point Address provides the highest level of accuracy for addresses.
- **Address Range:** The range of address points that are interpolated from the beginning and end of the street.
- **Geography:** Areas on a map that represent administrative divisions of a land, for example, country/region, state, or city.
- **POI:** Points on a map that are worth attention and that might be interesting.
- **Street:** Streets on the map. Addresses are resolved to the latitude and longitude coordinates of the street that contains the address. The house number might not be processed.
- **Cross Street:** Intersections. Cross streets represent junctions where two streets intersect.

Response

Let's look at the response structure. In the response that follows, the types of the result objects are different. If you look carefully, you see three types of result objects:

- Point Address
- Street
- Cross Street

Notice that the address search doesn't return POIs.

The `score` parameter for each response object indicates how the matching score relates to the scores of other objects in the same response. For more information about response object parameters, see [Get Search Address](#).

```
{
  "summary": {
    "query": "400 broad street seattle wa",
    "queryType": "NON_NEAR",
    "queryTime": 146,
    "numResults": 6,
    "offset": 0,
    "totalResults": 7,
    "fuzzyLevel": 1
  },
  "results": [
    {
      "type": "Point Address",
      "id": "US/PAD/p0/28725082",
      "score": 9.893799781799316,
      "address": {
        "streetNumber": "400",
        "streetName": "Broad Street",
      },
      "position": {
        "lat": 47.62039,
        "lon": -122.34928
      },
      "viewport": {
        "topLeftPoint": {
          "lat": 47.62129,
          "lon": -122.35061
        }
      }
    }
  ]
}
```

```
        "lon": -122.34795,
        "valid": true
    },
    "btmRightPoint": {
        "lat": 47.61949,
        "lon": -122.34795,
        "valid": true
    }
},
"entryPoints": [
    {
        "type": "main",
        "position": {
            "lat": 47.61982,
            "lon": -122.34886,
            "valid": true
        }
    }
]
},
{
    "type": "Street",
    "id": "US/STR/p0/6700384",
    "score": 8.129190444946289,
    "address": {
        "streetName": "Broad Street",
    },
    "position": {
        "lat": 47.61724,
        "lon": -122.35207
    },
    "viewport": {
        "topLeftPoint": {
            "lat": 47.61825,
            "lon": -122.35336,
            "valid": true
        },
        "btmRightPoint": {
            "lat": 47.61626,
            "lon": -122.35078,
            "valid": true
        }
    }
},
{
    "type": "Street",
    "id": "US/STR/p0/9701953",
    "score": 8.129190444946289,
    "address": {
        "streetName": "Broad Street",
    },
    "position": {
        "lat": 47.61965,
        "lon": -122.349
    },
    "viewport": {
        "topLeftPoint": {
            "lat": 47.62066,
            "lon": -122.35041,
            "valid": true
        },
        "btmRightPoint": {
            "lat": 47.61857,
            "lon": -122.34761,
            "valid": true
        }
    }
},
{
    "type": "Street"
}
```

```

    "type": "street",
    "id": "US/STR/p0/11721297",
    "score": 8.129190444946289,
    "address": {
        "streetName": "Broad Street",
        "municipalitySubdivision": "Seattle, Downtown Seattle, Denny Regrade, Belltown",
        "municipality": "Seattle",
        "countrySecondarySubdivision": "King",
        "countryTertiarySubdivision": "Seattle",
        "countrySubdivision": "WA",
        "countrySubdivisionName": "Washington",
        "postalCode": "98121",
        "extendedPostalCode": "981211237",
        "countryCode": "US",
        "country": "United States",
        "countryCodeISO3": "USA",
        "freeformAddress": "Broad Street, Seattle, WA 98121"
    },
    "position": {
        "lat": 47.61825,
        "lon": -122.35078
    },
    "viewport": {
        "topLeftPoint": {
            "lat": 47.61857,
            "lon": -122.35078,
            "valid": true
        },
        "btmRightPoint": {
            "lat": 47.61825,
            "lon": -122.35041,
            "valid": true
        }
    }
},
{
    "type": "Cross Street",
    "id": "US/XSTR/p1/232144",
    "score": 6.754479885101318,
    "address": {
        "streetName": "Broad Street & Valley Street",
        "municipalitySubdivision": "South Lake Union, Seattle",
    },
    "position": {
        "lat": 47.62545,
        "lon": -122.33974
    },
    "viewport": {
        "topLeftPoint": {
            "lat": 47.62635,
            "lon": -122.34107,
            "valid": true
        },
        "btmRightPoint": {
            "lat": 47.62455,
            "lon": -122.33841,
            "valid": true
        }
    }
}
]
}

```

Geometry

A response type of *Geometry* can include the geometry ID that's returned in the `dataSources` object under `geometry` and `id`. For example, you can use the [Search Polygon service](#) to request the geometry data in a

GeoJSON format. By using this format, you can get a city or airport outline for a set of entities. You can then use this boundary data to [Set up a geofence](#) or [Search POIs inside the geometry](#).

Responses for the [Search Address](#) API or the [Search Fuzzy](#) API can include the geometry ID that's returned in the `dataSources` object under `geometry` and `id`:

```
"dataSources": {  
    "geometry": {  
        "id": "00005557-4100-3c00-0000-000059690938" // The geometry ID is returned in the dataSources  
        object under "geometry" and "id".  
    }  
}
```

Next steps

To learn more, please see:

[How to build Azure Maps Search Service requests](#)

[Search Service API documentation](#)

Best practices for Azure Maps Route service

3/5/2021 • 11 minutes to read • [Edit Online](#)

The Route Directions and Route Matrix APIs in Azure Maps [Route Service](#) can be used to calculate the estimated arrival times (ETAs) for each requested route. Route APIs consider factors such as real-time traffic information and historic traffic data, like the typical road speeds on the requested day of the week and time of day. The APIs return the shortest or fastest routes available to multiple destinations at a time in sequence or in optimized order, based on time or distance. Users can also request specialized routes and details for walkers, bicyclists, and commercial vehicles like trucks. In this article, we'll share the best practices to call Azure Maps [Route Service](#), and you'll learn how-to:

- Choose between the Route Directions APIs and the Matrix Routing API
- Request historic and predicted travel times, based on real-time and historical traffic data
- Request route details, like time and distance, for the entire route and each leg of the route
- Request route for a commercial vehicle, like a truck
- Request traffic information along a route, like jams and toll information
- Request a route that consists of one or more stops (waypoints)
- Optimize a route of one or more stops to obtain the best order to visit each stop (waypoint)
- Optimize alternative routes using supporting points. For example, offer alternative routes that pass an electric vehicle charging station.
- Use the [Route Service](#) with the Azure Maps Web SDK

Prerequisites

1. [Make an Azure Maps account](#)
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.

For more information about the coverage of the Route Service, see the [Routing Coverage](#).

This article uses the [Postman app](#) to build REST calls, but you can choose any API development environment.

Choose between Route Directions and Matrix Routing

The Route Directions APIs return instructions including the travel time and the coordinates for a route path. The Route Matrix API lets you calculate the travel time and distances for a set of routes that are defined by origin and destination locations. For every given origin, the Matrix API calculates the cost (travel time and distance) of routing from that origin to every given destination. All of these APIs allow you to specify parameters such as the desired departure time, arrival times, and the vehicle type, like car or truck. They all use real-time or predictive traffic data accordingly to return the most optimal routes.

Consider calling Route Directions APIs if your scenario is to:

- Request the shortest or fastest driving route between two or more known locations, to get precise arrival times for your delivery vehicles.
- Request detailed route guidance, including route geometry, to visualize routes on the map
- Given a list of customer locations, calculate the shortest possible route to visit each customer location and return to the origin. This scenario is commonly known as the traveling salesman problem. You can pass up to 150 waypoints (stops) in one request.
- Send batches of queries to the Route Directions Batch API using just a single API call.

Consider calling Matrix Routing API if your scenario is to:

- Calculate the travel time or distance between a set of origins and destinations. For example, you have 12 drivers and you need to find the closest available driver to pick up the food delivery from the restaurant.
- Sort potential routes by their actual travel distance or time. The Matrix API returns only travel times and distances for each origin and destination combination.
- Cluster data based on travel time or distances. For example, your company has 50 employees, find all employees that live within 20 minute Drive Time from your office.

Here is a comparison to show some capabilities of the Route Directions and Matrix APIs:

AZURE MAPS API	MAX NUMBER OF QUERIES IN THE REQUEST	AVOID AREAS	TRUCK AND ELECTRIC VEHICLE ROUTING	WAYPOINTS AND TRAVELING SALESMAN OPTIMIZATION	SUPPORTING POINTS
Get Route Directions	1		✓	✓	
Post Route Directions	1	✓	✓	✓	✓
Post Route Directions Batch	700		✓	✓	
Post Route Matrix	700		✓		

To learn more about electric vehicle routing capabilities, see our tutorial on how to [route electric vehicles using Azure Notebooks with Python](#).

Request historic and real-time data

By default, the Route service assumes the traveling mode is a car and the departure time is now. It returns route based on real-time traffic conditions unless a route calculation request specifies otherwise. Fixed time-dependent traffic restrictions, like 'Left turns aren't allowed between 4:00 PM to 6:00 PM' are captured and will be considered by the routing engine. Road closures, like roadworks, will be considered unless you specifically request a route that ignores the current live traffic. To ignore the current traffic, set `traffic` to `false` in your API request.

The route calculation `travelTimeInSeconds` value includes the delay due to traffic. It's generated by leveraging the current and historic travel time data, when departure time is set to now. If your departure time is set in the future, the APIs return predicted travel times based on historical data.

If you include the `computeTravelTimeFor=all` parameter in your request, then the summary element in the response will have the following additional fields including historical traffic conditions:

ELEMENT	DESCRIPTION
<code>noTrafficTravelTimeInSeconds</code>	Estimated travel time calculated as if there are no delays on the route because of traffic conditions, for example, because of congestion
<code>historicTrafficTravelTimeInSeconds</code>	Estimated travel time calculated using time-dependent historic traffic data

ELEMENT	DESCRIPTION
liveTrafficIncidentsTravelTimeInSeconds	Estimated travel time calculated using real-time speed data

The next sections demonstrate how to make calls to the Route APIs using the discussed parameters.

Sample query

In the first example below the departure time is set to the future, at the time of writing.

```
https://atlas.microsoft.com/route/directions/json?subscription-key=<Your-Azure-Maps-Primary-Subscription-Key>&api-version=1.0&query=51.368752,-0.118332:51.385426,-0.128929&travelMode=car&traffic=true&departAt=2025-03-29T08:00:20&computeTravelTimeFor=all
```

The response contains a summary element, like the one below. Because the departure time is set to the future, the **trafficDelayInSeconds** value is zero. The **travelTimeInSeconds** value is calculated using time-dependent historic traffic data. So, in this case, the **travelTimeInSeconds** value is equal to the **historicTrafficTravelTimeInSeconds** value.

```
"summary": {
    "lengthInMeters": 2131,
    "travelTimeInSeconds": 248,
    "trafficDelayInSeconds": 0,
    "departureTime": "2025-03-29T08:00:20Z",
    "arrivalTime": "2025-03-29T08:04:28Z",
    "noTrafficTravelTimeInSeconds": 225,
    "historicTrafficTravelTimeInSeconds": 248,
    "liveTrafficIncidentsTravelTimeInSeconds": 248
},
```

Sample query

In the second example below, we have a real-time routing request, where departure time is now. It's not explicitly specified in the URL because it's the default value.

```
https://atlas.microsoft.com/route/directions/json?subscription-key=<Your-Azure-Maps-Primary-Subscription-Key>&api-version=1.0&query=47.6422356,-122.1389797:47.6641142,-122.3011268&travelMode=car&traffic=true&computeTravelTimeFor=all
```

The response contains a summary as shown below. Because of congestions, the **trafficDelaysInSeconds** value is greater than zero. It's also greater than **historicTrafficTravelTimeInSeconds**.

```
"summary": {
    "lengthInMeters": 16637,
    "travelTimeInSeconds": 2905,
    "trafficDelayInSeconds": 1604,
    "departureTime": "2020-02-28T01:00:20+00:00",
    "arrivalTime": "2020-02-28T01:48:45+00:00",
    "noTrafficTravelTimeInSeconds": 872,
    "historicTrafficTravelTimeInSeconds": 1976,
    "liveTrafficIncidentsTravelTimeInSeconds": 2905
},
```

Request route and leg details

By default, the Route service will return an array of coordinates. The response will contain the coordinates that make up the path in a list named `points`. Route response also includes the distance from the start of the route

and the estimated elapsed time. These values can be used to calculate the average speed for the entire route.

The following image shows the `points` element.

```
"routes": [
  {
    "summary": {
      "lengthInMeters": 2131,
      "travelTimeInSeconds": 248,
      "trafficDelayInSeconds": 0,
      "departureTime": "2025-03-29T08:00:20Z",
      "arrivalTime": "2025-03-29T08:04:28Z",
      "noTrafficTravelTimeInSeconds": 225,
      "historicTrafficTravelTimeInSeconds": 248,
      "liveTrafficIncidentsTravelTimeInSeconds": 248
    },
    "legs": [
      {
        "summary": {
          "lengthInMeters": 2131,
          "travelTimeInSeconds": 248,
          "trafficDelayInSeconds": 0,
          "departureTime": "2025-03-29T08:00:20Z",
          "arrivalTime": "2025-03-29T08:04:28Z",
          "noTrafficTravelTimeInSeconds": 225,
          "historicTrafficTravelTimeInSeconds": 248,
          "liveTrafficIncidentsTravelTimeInSeconds": 248
        },
        "points": [ ... //list of 127 points
      ]
    ]
  ],
}
```

Expand the `point` element to see the list of coordinates for the path:

```
"points": [
  {
    "latitude": 51.36875,
    "longitude": -0.11833
  },
  {
    "latitude": 51.36882,
    "longitude": -0.11827
  },
  {
    "latitude": 51.36895,
    "longitude": -0.11815
  },
  {
    "latitude": 51.36915,
    "longitude": -0.11798
  },
  {
    "latitude": 51.36989,
    "longitude": -0.11735
  },
  ...
]
```

The Route Directions APIs support different formats of instructions that can be used by specifying the `instructionsType` parameter. To format instructions for easy computer processing, use `instructionsType=coded`. Use `instructionsType=tagged` to display instructions as text for the user. Also, instructions can be formatted as text where some elements of the instructions are marked, and the instruction is presented with special formatting. For more information, see the [list of supported instruction types](#).

When instructions are requested, the response returns a new element named `guidance`. The `guidance` element holds two pieces of information: turn-by-turn directions and summarized instructions.

```
{  
    "formatVersion": "0.0.12",  
    "routes": [  
        {  
            "summary": { ... },  
            "legs": [ ... ],  
            "sections": [ ... ],  
            "guidance": {  
                "instructions": [ ... //11 items ],  
                "instructionGroups": [ ... //3 items ]  
            }  
        }  
    ]  
}
```

The `instructions` element holds turn-by-turn directions for the trip, and the `instructionGroups` has summarized instructions. Each instruction summary covers a segment of the trip that could cover multiple roads. The APIs can return details for sections of a route, such as, the coordinate range of a traffic jam or the current speed of traffic.

```
"instructions": [  
    {  
        "routeOffsetInMeters": 0,  
        "travelTimeInSeconds": 0,  
        "point": { ... },  
        "instructionType": "LOCATION_DEPARTURE",  
        "roadNumbers": [ ... ],  
        "street": "Purley Way",  
        "possibleCombineWithNext": false,  
        "drivingSide": "LEFT",  
        "maneuver": "DEPART",  
        "message": "Leave from <street>Purley Way</street>/<roadNumber>A23</roadNumber>"  
    },  
    {  
        "routeOffsetInMeters": 1206,  
        "travelTimeInSeconds": 116,  
        "point": {  
            "latitude": 51.37893,  
            "longitude": -0.12118  
        },  
        "instructionType": "TURN",  
        "street": "Ampere Way",  
        "junctionType": "REGULAR",  
        "turnAngleInDecimalDegrees": -45,  
        "possibleCombineWithNext": false,  
        "drivingSide": "LEFT",  
        "maneuver": "BEAR_LEFT",  
        "message": "Bear left at <street>Ampere Way</street>"  
    },  
  
    "instructionGroups": [  
        {  
            "firstInstructionIndex": 0,  
            "lastInstructionIndex": 4,  
            "groupMessage": "Leave from <street>Purley Way</street>. Take the <roadNumber>A23</roadNumber>. Continue to your destination at <street>Ampere Way</street>",  
            "groupLengthInMeters": 2131  
        }  
    ]  
]
```

Request a route for a commercial vehicle

Azure Maps Routing APIs support commercial vehicle routing, covering commercial trucks routing. The APIs consider specified limits. Such as, the height and weight of the vehicle, and if the vehicle is carrying hazardous cargo. For example, if a vehicle is carrying flammable, the routing engine avoid certain tunnels that are near residential areas.

Sample query

The sample request below queries a route for a commercial truck. The truck is carrying class 1 hazardous waste material.

```
https://atlas.microsoft.com/route/directions/json?subscription-key=<Your-Azure-Maps-Primary-Subscription-Key>&api-version=1.0&vehicleWidth=2&vehicleHeight=2&vehicleCommercial=true&vehicleLoadType=USHazmatClass1&travelMode=truck&instructionsType=text&query=51.368752,-0.118332:41.385426,-0.128929
```

The Route API returns directions that accommodate the dimensions of the truck and the hazardous waste. You can read the route instructions by expanding the `guidance` element.

The screenshot shows a JSON response with the 'guidance' element expanded. The 'guidance' object contains two arrays: 'instructions' and 'instructionGroups'. The 'instructions' array is shown as a list of items, and the 'instructionGroups' array is shown as a list containing 8 items. The rest of the JSON structure is collapsed, indicated by ellipses (...).

```
{
  "formatVersion": "0.0.12",
  "routes": [
    {
      "summary": {
        "lengthInMeters": 1506890,
        "travelTimeInSeconds": 66542,
        "trafficDelayInSeconds": 197,
        "departureTime": "2020-03-11T21:22:25Z",
        "arrivalTime": "2020-03-12T15:51:27+00:00"
      },
      "legs": [...],
      "sections": [...],
      "guidance": {
        "instructions": [...],
        "instructionGroups": [...] //contains 8 items
      }
    }
  ]
}
```

Sample query

Changing the US Hazmat Class, from the above query, will result in a different route to accommodate this change.

```
https://atlas.microsoft.com/route/directions/json?subscription-key=<Your-Azure-Maps-Primary-Subscription-Key>&api-version=1.0&vehicleWidth=2&vehicleHeight=2&vehicleCommercial=true&vehicleLoadType=USHazmatClass9&travelMode=truck&instructionsType=text&query=51.368752,-0.118332:41.385426,-0.128929
```

The response below is for a truck carrying a class 9 hazardous material, which is less dangerous than a class 1 hazardous material. When you expand the `guidance` element to read the directions, you'll notice that the directions aren't the same. There are more route instructions for the truck carrying class 1 hazardous material.

```
{  
    "formatVersion": "0.0.12",  
    "routes": [  
        {  
            "summary": {  
                "lengthInMeters": 1506890,  
                "travelTimeInSeconds": 66512,  
                "trafficDelayInSeconds": 183,  
                "departureTime": "2020-03-11T21:26:04Z",  
                "arrivalTime": "2020-03-12T15:54:35+00:00"  
            },  
            "legs": [...  
            ],  
            "sections": [...  
            ],  
            "guidance": {  
                "instructions": [...  
                ],  
                "instructionGroups": [...] //contains 7 items  
            }  
        }  
    ]  
}
```

Request traffic information along a route

With the Azure Maps Route Direction APIs, developers can request details for each section type by including the `sectionType` parameter in the request. For example, you can request the speed information for each traffic jam segment. Refer to the [list of values for the sectionType key](#) to learn about the various details that you can request.

Sample query

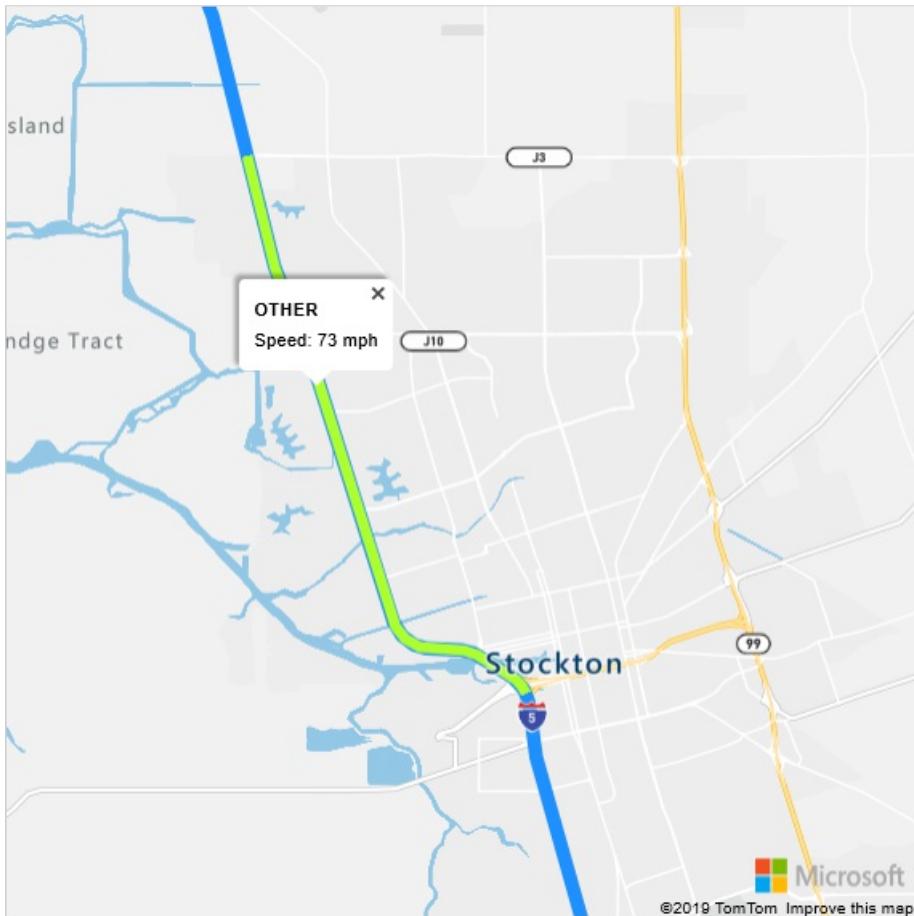
The following query sets the `sectionType` to `traffic`. It requests the sections that contain traffic information from Seattle to San Diego.

```
https://atlas.microsoft.com/route/directions/json?subscription-key=<Your-Azure-Maps-Primary-Subscription-Key>&api-version=1.0&sectionType=traffic&query=47.6062,-122.3321:32.7157,-117.1611
```

The response contains the sections that are suitable for traffic along the given coordinates.

```
{
  "formatVersion": "0.0.12",
  "routes": [
    {
      "summary": {
        "lengthInMeters": 2077897,
        "travelTimeInSeconds": 71160,
        "trafficDelayInSeconds": 0,
        "departureTime": "2020-03-11T21:58:52+00:00",
        "arrivalTime": "2020-03-12T17:44:52+00:00"
      },
      "legs": [...],
      "sections": [
        {
          "startPointIndex": 2993,
          "endPointIndex": 3064,
          "sectionType": "TRAFFIC",
          "simpleCategory": "JAM",
          "effectiveSpeedInKmh": 22,
          "delayInSeconds": 0,
          "magnitudeOfDelay": 2,
          "tec": {
            "causes": [
              {
                "mainCauseCode": 1
              }
            ],
            "effectCode": 6
          }
        }
      ]
    }
  ]
}
```

This option can be used to color the sections when rendering the map, as in the image below:



Calculate and optimize a multi-stop route

Azure Maps currently provides two forms of route optimizations:

- Optimizations based on the requested route type, without changing the order of waypoints. You can find the [supported route types here](#)
- Traveling salesman optimization, which changes the order of the waypoints to obtain the best order to visit each stop

For multi-stop routing, up to 150 waypoints may be specified in a single route request. The starting and ending coordinate locations can be the same, as would be the case with a round trip. But you need to provide at least one additional waypoint to make the route calculation. Waypoints can be added to the query in-between the origin and destination coordinates.

If you want to optimize the best order to visit the given waypoints, then you need to specify `computeBestOrder=true`. This scenario is also known as the traveling salesman optimization problem.

Sample query

The following query requests the path for six waypoints, with the `computeBestOrder` parameter set to `false`. It's also the default value for the `computeBestOrder` parameter.

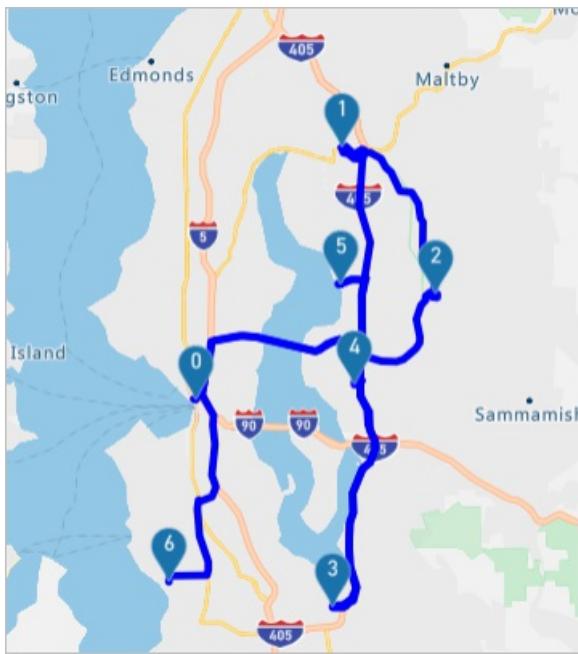
```
https://atlas.microsoft.com/route/directions/json?api-version=1.0&subscription-key=<Your-Azure-Maps-Primary-Subscription-Key>&computeBestOrder=false&query=47.606544,-122.336502:47.759892,-122.204821:47.670682,-122.120415:47.480133,-122.213369:47.615556,-122.193689:47.676508,-122.206054:47.495472,-122.360861
```

The response describes the path length to be 140,851 meters, and that it would take 9,991 seconds to travel that path.



```
{
  "formatVersion": "0.0.12",
  "routes": [
    {
      "summary": {
        "lengthInMeters": 140851,
        "travelTimeInSeconds": 9991,
        "trafficDelayInSeconds": 288,
        "departureTime": "2020-03-12T00:27:02+00:00",
        "arrivalTime": "2020-03-12T03:13:31+00:00"
      },
      "legs": [ ... ],
      "sections": [ ... ]
    }
  ]
}
```

The image below illustrates the path resulting from this query. This path is one possible route. It's not the optimal path based on time or distance.



This route waypoint order is: 0, 1, 2, 3, 4, 5, and 6.

Sample query

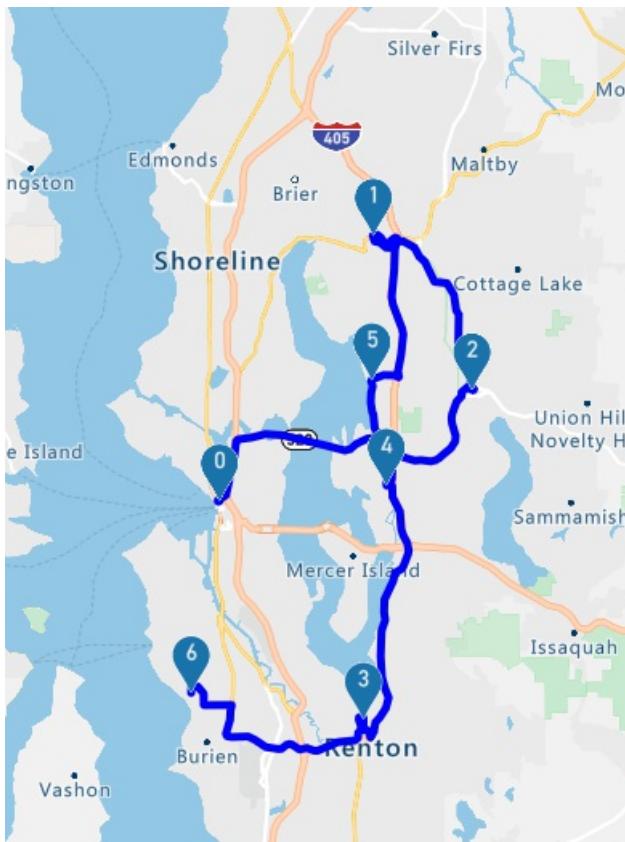
The following query requests the path for the same six waypoints, as in the above sample. This time, the `computeBestOrder` parameter set to `true` (the traveling salesman optimization).

```
https://atlas.microsoft.com/route/directions/json?api-version=1.0&subscription-key=<Your-Azure-Maps-Primary-Subscription-Key>&computeBestOrder=true&query=47.606544,-122.336502:47.759892,-122.204821:47.670682,-122.120415:47.480133,-122.213369:47.615556,-122.193689:47.676508,-122.206054:47.495472,-122.360861
```

The response describes the path length to be 91,814 meters, and that it would take 7,797 seconds to travel that path. The travel distance and the travel time are both lower here because the API returned the optimized route.

```
[{"formatVersion": "0.0.12", "routes": [ { "summary": { "lengthInMeters": 91814, "travelTimeInSeconds": 7797, "trafficDelayInSeconds": 314, "departureTime": "2020-03-12T00:38:28+00:00", "arrivalTime": "2020-03-12T02:48:23+00:00" }, "legs": [ ... ], "sections": [ ... ] } ], "optimizedWaypoints": [ ... ] }
```

The image below illustrates the path resulting from this query.



The optimal route has the following waypoint order: 0, 5, 1, 2, 4, 3, and 6.

TIP

The optimized waypoint order information from the Routing service provides a set of indices. These exclude the origin and the destination indices. You need to increment these values by 1 to account for the origin. Then, add your destination to the end to get the complete ordered waypoint list.

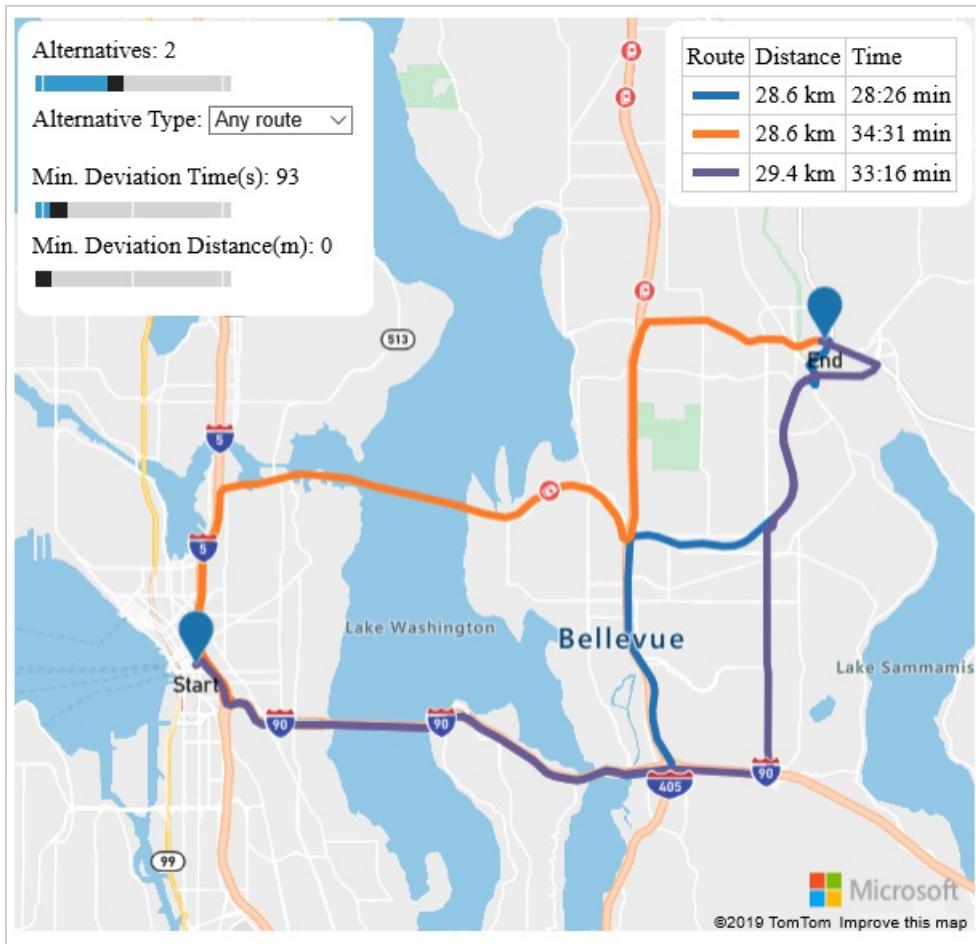
Calculate and bias alternative routes using supporting points

You might have situations where you want to reconstruct a route to calculate zero or more alternative routes for a reference route. For example, you may want to show customers alternative routes that pass your retail store. In this case, you need to bias a location using supporting points. Here are the steps to bias a location:

1. Calculate a route as-is and get the path from the route response
2. Use the route path to find the desired locations along or near the route path. For example, you can use Azure Maps [Point of Interest API](#) or query your own data in your database.
3. Order the locations based on the distance from the start of the route
4. Add these locations as supporting points in a new route request to the [Post Route Directions API](#). To learn more about the supporting points, see the [Post Route Directions API documentation](#).

When calling the [Post Route Directions API](#), you can set the minimum deviation time or the distance constraints, along with the supporting points. Use these parameters if you want to offer alternative routes, but you also want to limit the travel time. When these constraints are used, the alternative routes will follow the reference route from the origin point for the given time or distance. In other words, the other routes diverge from the reference route per the given constraints.

The image below is an example of rendering alternative routes with specified deviation limits for the time and the distance.



Use the Routing service in a web app

The Azure Maps Web SDK provides a [Service module](#). This module is a helper library that makes it easy to use the Azure Maps REST APIs in web or Node.js applications, using JavaScript or TypeScript. The Service module can be used to render the returned routes on the map. The module automatically determines which API to use with GET and POST requests.

Next steps

To learn more, please see:

[Azure Maps Route service](#)

[How to use the Service module](#)

[Show route on the map](#)

[Azure Maps NPM Package](#)

Render custom data on a raster map

6/1/2021 • 5 minutes to read • [Edit Online](#)

This article explains how to use the [static image service](#), with image composition functionality, to allow overlays on top of a raster map. Image composition includes the ability to get a raster tile back, with additional data like custom pushpins, labels, and geometry overlays.

To render custom pushpins, labels, and geometry overlays, you can use the Postman application. You can use Azure Maps [Data Service APIs](#) to store and render overlays.

TIP

To show a simple map on a web page, it's often more cost effective to use the Azure Maps Web SDK, rather than to use the static image service. The web SDK uses map tiles; and unless the user pans and zooms the map, they will often generate only a fraction of a transaction per map load. The Azure Maps web SDK has options for disabling panning and zooming. Additionally, the Azure Maps web SDK provides a richer set of data visualization options than a static map web service does.

Prerequisites

1. [Make an Azure Maps account](#)
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.

This tutorial uses the [Postman](#) application, but you may use a different API development environment.

Render pushpins with labels and a custom image

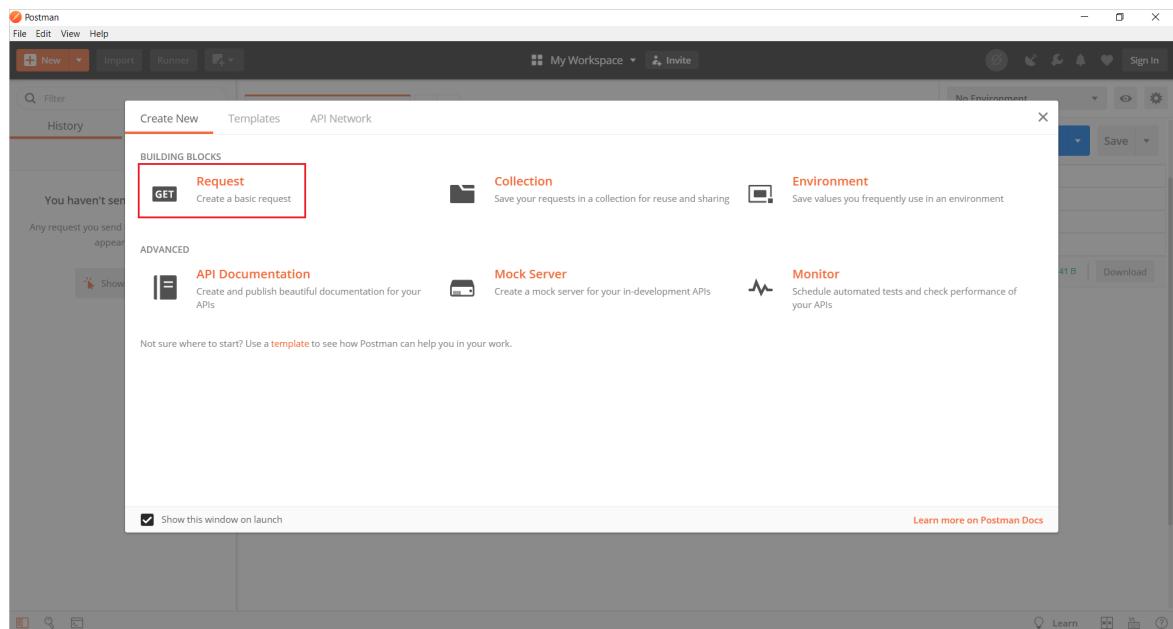
NOTE

The procedure in this section requires an Azure Maps account in Gen 1 or Gen 2 pricing tier.

The Azure Maps account Gen 1 Standard S0 tier supports only a single instance of the `pins` parameter. It allows you to render up to five pushpins, specified in the URL request, with a custom image.

To render pushpins with labels and a custom image, complete these steps:

1. Create a collection in which to store the requests. In the Postman app, select **New**. In the **Create New** window, select **Collection**. Name the collection and select the **Create** button.
2. To create the request, select **New** again. In the **Create New** window, select **Request**. Enter a **Request name** for the pushpins. Select the collection you created in the previous step, as the location to save the request. Then, select **Save**.



3. Select the GET HTTP method on the builder tab and enter the following URL to create a GET request.

```
https://atlas.microsoft.com/map/static/png?subscription-key={subscription-key}&api-version=1.0&layer=basic&style=main&zoom=12&center=-73.98,%2040.77&pins=custom%7Cla15+50%7C1s12%7C1c003b61%7C%7C%27CentralPark%27-73.9657974+40.781971%7C%7Chttps%3A%2F%2Fraw.githubusercontent.com%2FAzure-Samples%2FAzureMapsCodeSamples%2Fmaster%2FAzureMapsCodeSamples%2FCommon%2Fimages%2Ficons%2Fyellow-pushpin.png
```

Here's the resulting image:



Get data from Azure Maps data storage

NOTE

The procedure in this section requires an Azure Maps account Gen 1 (S1) or Gen 2 pricing tier.

You can also obtain the path and pin location information by using the [Data Upload API](#). Follow the steps below to upload the path and pins data.

1. In the Postman app, open a new tab in the collection you created in the previous section. Select the POST HTTP method on the builder tab and enter the following URL to make a POST request:

```
https://us.atlas.microsoft.com/mapData?subscription-key={subscription-key}&api-version=2.0&dataFormat=geojson
```

2. On the **Params** tab, enter the following key/value pairs, which are used for the POST request URL. Replace the `subscription-key` value with your Azure Maps subscription key.

POST https://atlas.microsoft.com/mapData/upload?subscription-key={subscription-key}&api-version=1.0&dataFormat=geojson

Params (1) Authorization Headers (1) Body Pre-request Script Tests Cookies Code Comments (0)

KEY	VALUE	DESCRIPTION
subscription-key	{subscription-key}	
apiVersion	1.0	
dataFormat	geojson	

Body Cookies Headers (7) Test Results

Transfer-Encoding → chunked
Content-Type → application/json
Location → https://atlas.microsoft.com/mapData/<uuid>/status?api-version=1.0&subscription-key=<subscription-key>
Access-Control-Expose-Headers → Location
X-Correlation-ID → bf622c6-5c6d-4941-987d-89c7f9006bc3
x-content-type-options → nosniff
Date → Thu, 31 Jan 2019 20:33:51 GMT

Status: 202 Accepted Time: 2981 ms Size: 441 B Download

3. On the **Body** tab, select the raw input format and choose JSON as the input format from the dropdown list. Provide this JSON as data to be uploaded:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [
              [
                [
                  [
                    [
                      [
                        [
                          [
                            [
                              [
                                [
                                  [
                                    [
                                      [
                                        [
                                          [
                                            [
                                              [
                                                [
                                                  [
                                                    [
                                                      [
                                                        [
                                                          [
                                                            [
                                                              [
                                                                [
                                                                  [
                                                                    [
                                                                      [
                                                                        [
                                                                          [
                                                                            [
                                                                              [
                                                                                [
                                                                                  [
                                                                                    [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
                                                                                      [
................................................................
```

4. Select **Send** and review the response header. Upon a successful request, the *Operation-Location* header will contain the `status URL` to check the current status of the upload request. The `status URL` has the following format:

```
https://us.atlas.microsoft.com/mapData/operations/{statusUrl}?api-version=2.0
```

5. Copy your status URI and append the subscription-key parameter to it with the value of your Azure Maps account subscription key. Use the same account subscription key that you used to upload the data. The status URI format should look like the one below:

```
https://us.atlas.microsoft.com/mapData/operations/{statusUrl}?api-version=2.0&subscription-key={Subscription-key}
```

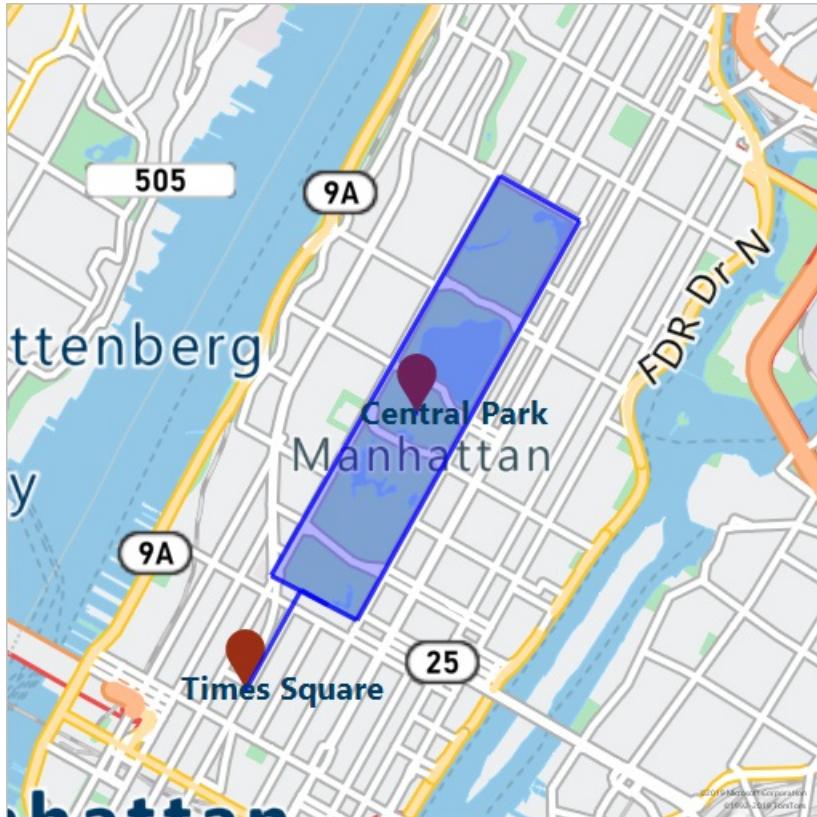
6. To get the `udid`, open a new tab in the Postman app. Select GET HTTP method on the builder tab. Make a GET request at the `status URL`. If your data upload was successful, you'll receive a `udid` in the response body. Copy the `udid`.

```
{  
    "udid" : "{udId}"  
}
```

7. Use the `udid` value received from the Data Upload API to render features on the map. To do so, open a new tab in the collection you created in the preceding section. Select the GET HTTP method on the builder tab, replace the `{subscription-key}` and `{udId}` with your values, and enter this URL to make a GET request:

```
https://atlas.microsoft.com/map/static/png?subscription-key={subscription-key}&api-version=1.0&layer=basic&style=main&zoom=12&center=-73.96682739257812%2C40.78119135317995&pins=default|la-35+50|ls12|lc003C62|co9B2F15||'Times Square'-73.98516297340393 40.758781646381024||'Central Park'-73.96682739257812 40.78119135317995&path=lc0000FF|fc0000FF|lw3|la0.80|fa0.30||udid-{udId}
```

Here's the response image:



Render a polygon with color and opacity

NOTE

The procedure in this section requires an Azure Maps account Gen 1 (S1) or Gen 2 pricing tier.

You can modify the appearance of a polygon by using style modifiers with the [path parameter](#).

1. In the Postman app, open a new tab in the collection you created earlier. Select the GET HTTP method on the builder tab and enter the following URL to configure a GET request to render a polygon with color and opacity:

```
https://atlas.microsoft.com/map/static/png?api-version=1.0&style=main&layer=basic&sku=S1&zoom=14&height=500&width=500&center=-74.040701,40.698666&path=lc0000FF|fc0000FF|lw3|la0.80|fa0.50|-74.03995513916016 40.70090237454063| -74.04082417488098 40.70028420372218|-74.04113531112671 40.70049568385827|-74.04298067092896 40.69899904076542|-74.04271245002747 40.69879568992435|-74.04367804527283 40.6980961582905|-74.04364585876465 40.698055487620714|-74.04368877410889 40.698022951066996|-74.04168248176573 40.696444909137|-74.03901100158691 40.69837271818651|-74.03824925422668 40.69837271818651|-74.03809905052185 40.69903971085914|-74.03771281242369 40.699340668780984|-74.03940796852112 40.70058515602143|-74.03948307037354 40.70052821920425|-74.03995513916016 40.70090237454063 &subscription-key={subscription-key}
```

Here's the response image:



Render a circle and pushpins with custom labels

NOTE

The procedure in this section requires an Azure Maps account Gen 1 (S1) or Gen 2 pricing tier.

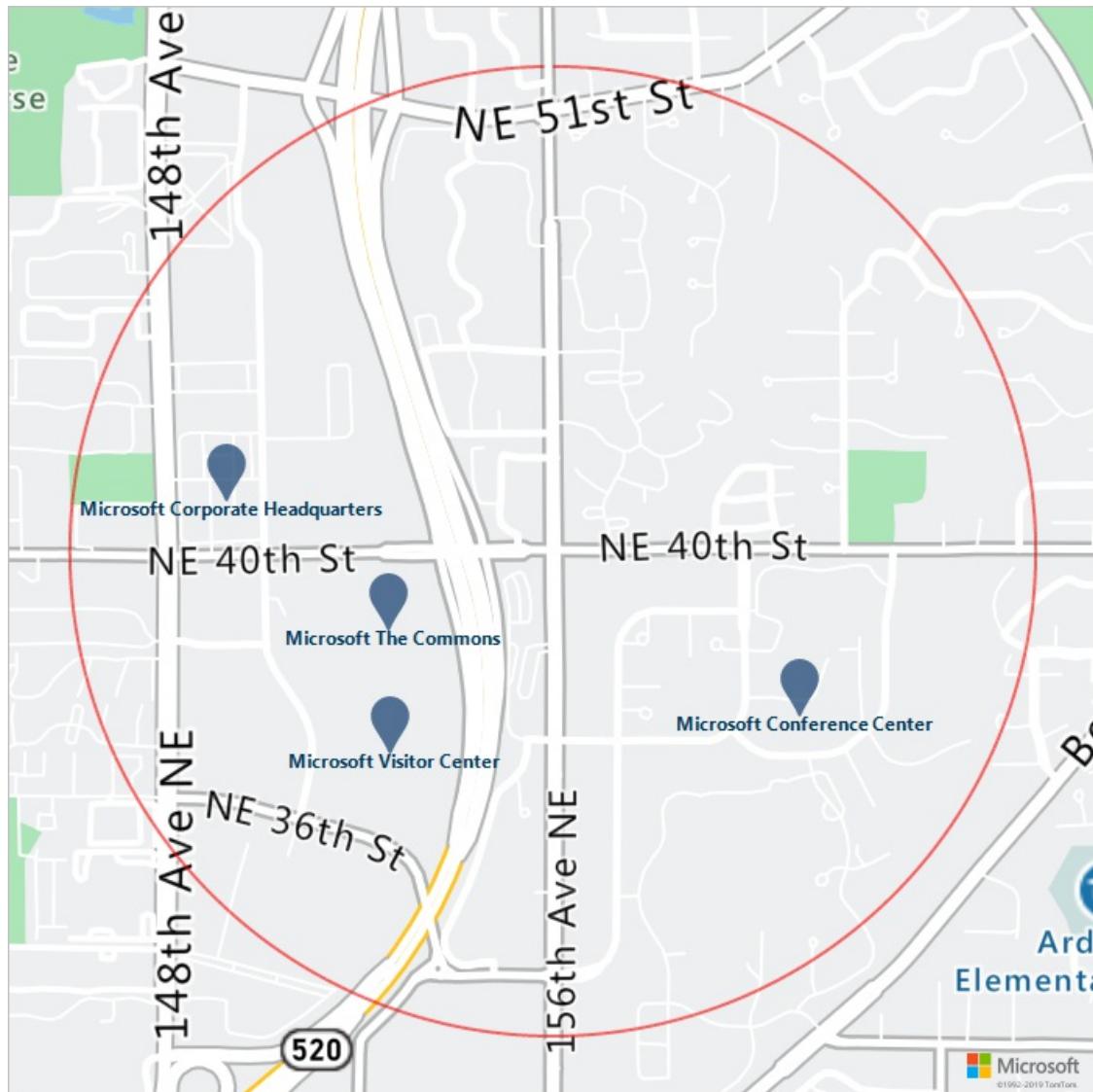
You can modify the appearance of the pins by adding style modifiers. For example, to make pushpins and their labels larger or smaller, use the `sc` "scale style" modifier. This modifier takes a value that's greater than zero. A value of 1 is the standard scale. Values larger than 1 will make the pins larger, and values smaller than 1 will make them smaller. For more information about style modifiers, see [static image service path parameters](#).

Follow these steps to render a circle and pushpins with custom labels:

1. In the Postman app, open a new tab in the collection you created earlier. Select the GET HTTP method on the builder tab and enter this URL to make a GET request:

```
https://atlas.microsoft.com/map/static/png?api-version=1.0&style=main&layer=basic&zoom=14&height=700&width=700&center=-122.13230609893799,47.64599069048016&path=lcff0000|lw2|la0.60|ra1000||-122.13230609893799|47.64599069048016&pins=default|la15+50|al0.66|lc003c62|co002d62||'Microsoft Corporate Headquarters'-122.14131832122801|47.64690503939462||'Microsoft Visitor Center'-122.136828|47.642224||'Microsoft Conference Center'-122.12552547454833|47.642940335653996||'Microsoft The Commons'-122.13687658309935|47.64452336193245&subscription-key={subscription-key}
```

Here's the response image:

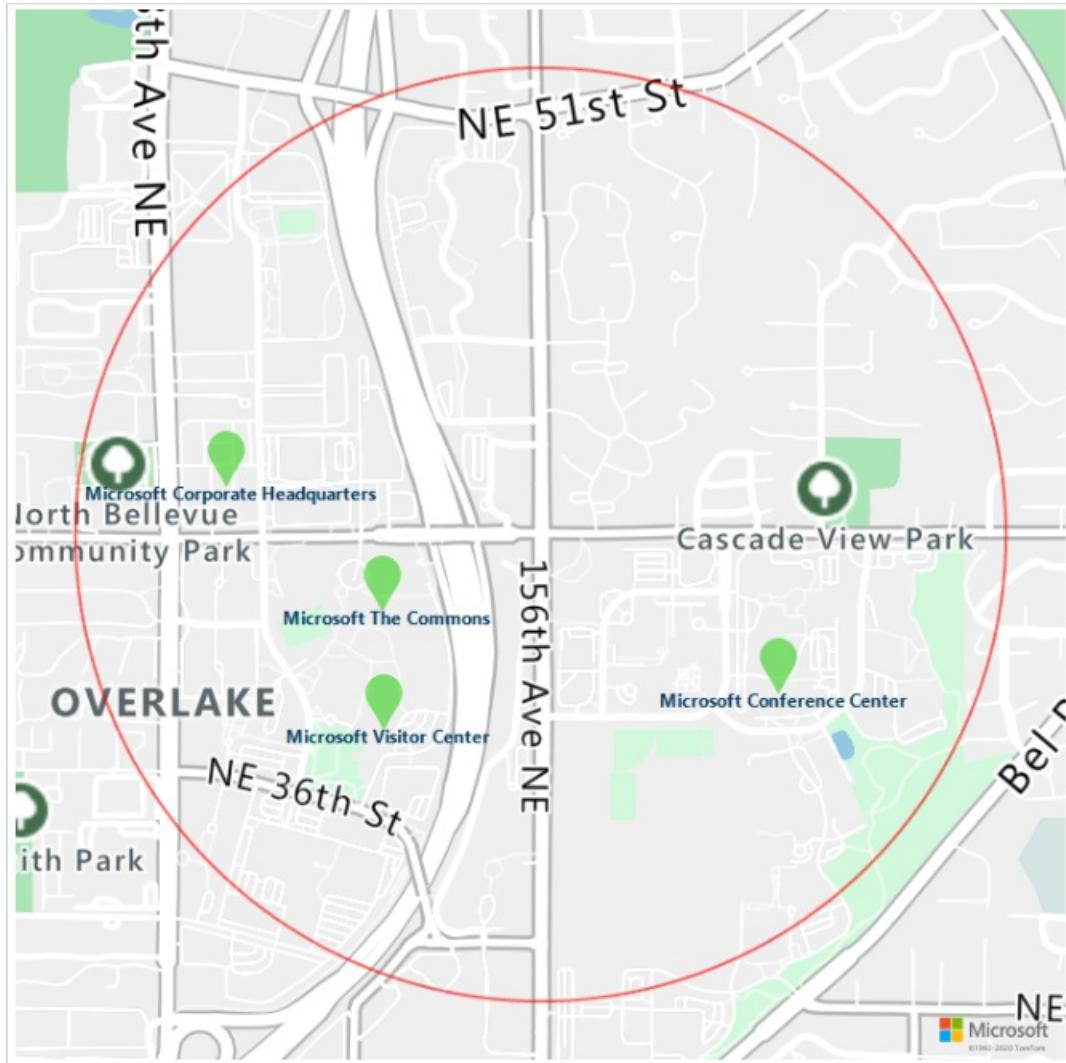


2. To change the color of the pushpins from the last step, change the "co" style modifier. Look at

`pins=default|1a15+50|a10.66|1c003C62|co002D62|`, the current color would be specified as #002D62 in CSS. Let's say you want to change it to #41d42a. Write the new color value after the "co" specifier, like this: `pins=default|1a15+50|a10.66|1c003C62|co41D42A|`. Make a new GET request:

```
https://atlas.microsoft.com/map/static/png?api-version=1.0&style=main&layer=basic&zoom=14&height=700&width=700&center=-122.13230609893799,47.64599069048016&path=lcff0000|lw2|1a0.60|ra1000||-122.13230609893799|47.64599069048016&pins=default|1a15+50|a10.66|1c003C62|co41D42A||'Microsoft Corporate Headquarters'-122.14131832122801 47.64690503939462||'Microsoft Visitor Center'-122.136828 47.642224||'Microsoft Conference Center'-122.12552547454833 47.642940335653996||'Microsoft The Commons'-122.13687658309935 47.64452336193245&subscription-key={subscription-key}
```

Here's the response image after changing the colors of the pins:



Similarly, you can change, add, and remove other style modifiers.

Next steps

- Explore the [Azure Maps Get Map Image API documentation](#).
- To learn more about Azure Maps Data service, see the [service documentation](#).

Request public transit data using the Azure Maps Mobility services (Preview)

3/5/2021 • 11 minutes to read • [Edit Online](#)

IMPORTANT

Azure Maps Mobility services are currently in public preview. This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

This article shows you how to use Azure Maps [Mobility services](#) to request public transit data. Transit data includes transit stops, route information, and travel time estimations.

In this article you'll learn, how to:

- Get a metro area ID using the [Get Metro Area API](#)
- Request nearby transit stops using [Get Nearby Transit service](#).
- Query [Get Transit Routes API](#) to plan a route using public transit.
- Request transit route geometry and detailed schedule for the route using the [Get Transit Itinerary API](#).

Prerequisites

1. [Make an Azure Maps account](#)
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

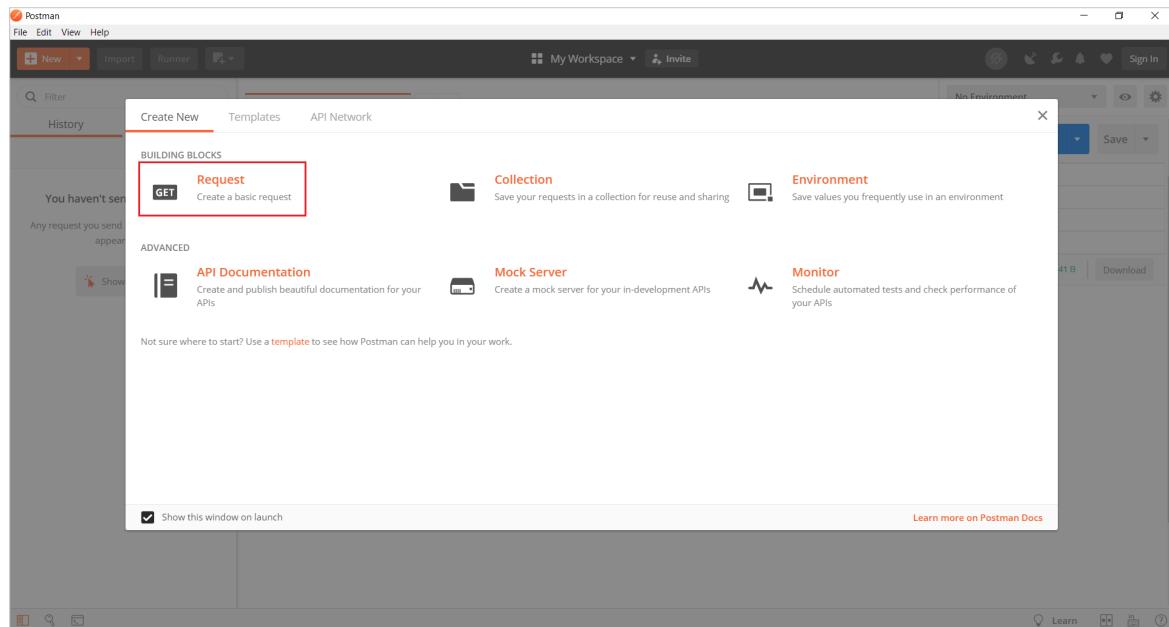
This tutorial uses the [Postman](#) application, but you may choose a different API development environment.

Get a metro area ID

In order to request detail information about transit agencies and supported transit types for a particular metropolitan area, you'll need the `metroId` of that area. The [Get Metro Area API](#) allows you to request metro areas, in which the Azure Maps Mobility services are available. The response includes details such as the `metroId`, `metroName`, and the representation of the metro area geometry in GeoJSON format.

Let's make a request to get the Metro Area for the Seattle-Tacoma metro area ID. To request ID for a metro area, complete the following steps:

1. Open the Postman app, and let's create a collection to store the requests. Near the top of the Postman app, select **New**. In the **Create New** window, select **Collection**. Name the collection and select the **Create** button.
2. To create the request, select **New** again. In the **Create New** window, select **Request**. Enter a **Request name** for the request. Select the collection you created in the previous step as the location in which to save the request. Then, select **Save**.



3. Select the **GET** HTTP method on the builder tab and enter the following URL to create a GET request.
Replace `{subscription-key}`, with your Azure Maps primary key.

```
https://atlas.microsoft.com/mobility.metroArea/id/json?subscription-key={subscription-key}&api-version=1.0&query=47.63096,-122.126
```

4. After a successful request, you'll receive the following response:

```
{  
    "results": [  
        {  
            "metroId": 522,  
            "metroName": "Seattle-Tacoma-Bellevue, WA",  
            "viewport": {  
                "topLeftPoint": {  
                    "latitude": 48.5853,  
                    "longitude": -124.80934  
                },  
                "btmRightPoint": {  
                    "latitude": 46.90534,  
                    "longitude": -121.55032  
                }  
            },  
            "geometry": {  
                "type": "Polygon",  
                "coordinates": [  
                    [  
                        [  
                            [  
                                [-121.99604,  
                                 47.16147  
                            ],  
                            [  
                                [-121.97051,  
                                 47.17222  
                            ],  
                            [  
                                [-121.96308,  
                                 47.17671  
                            ],  
                            ...  
                            ...  
                            ...  
                            [  
                                [-122.01525,  
                                 47.16008  
                            ],  
                            [  
                                [-122.00553,  
                                 47.15919  
                            ],  
                            [  
                                [-121.99604,  
                                 47.16147  
                            ]  
                        ]  
                    ]  
                ]  
            }  
        ]  
    ]  
}
```

Request nearby transit stops

The Azure Maps [Get Nearby Transit](#) service allows you to search transit objects. The API returns the transit object details, such as public transit stops and shared bikes around a given location. Next, we'll make a request to the service to search for nearby public transit stops within a 300-meters radius around the given location.

To make a request to the [Get Nearby Transit](#), follow the steps below:

1. In Postman, click **New Request | GET request** and name it **Get Nearby stops**.
2. On the Builder tab, select the **GET** HTTP method, enter the following request URL for your API endpoint

and click **Send**.

```
https://atlas.microsoft.com/mobility/transit/nearby/json?subscription-key={subscription-key}&api-version=1.0&query=47.63096,-122.126&radius=300&objectType=stop
```

3. After a successful request, the response structure should look like the one below:

```
{  
    "results": [  
        {  
            "id": "522---2060603",  
            "type": "stop",  
            "objectDetails": {  
                "stopKey": "71300",  
                "stopName": "NE 24th St & 162nd Ave NE",  
                "stopCode": "71300",  
                "mainTransitType": "Bus",  
                "mainAgencyId": "522---5872",  
                "mainAgencyName": "Metro Transit"  
            },  
            "position": {  
                "latitude": 47.631504,  
                "longitude": -122.125275  
            },  
            "viewport": {  
                "topLeftPoint": {  
                    "latitude": 47.632413,  
                    "longitude": -122.12659  
                },  
                "btmRightPoint": {  
                    "latitude": 47.630594,  
                    "longitude": -122.123959  
                }  
            }  
        },  
        {  
            "id": "522---2061020",  
            "type": "stop",  
            "objectDetails": {  
                "stopKey": "68372",  
                "stopName": "NE 24th St & 160th Ave NE",  
                "stopCode": "68372",  
                "mainTransitType": "Bus",  
                "mainAgencyId": "522---5872",  
                "mainAgencyName": "Metro Transit"  
            },  
            "position": {  
                "latitude": 47.631409,  
                "longitude": -122.127136  
            },  
            "viewport": {  
                "topLeftPoint": {  
                    "latitude": 47.632318,  
                    "longitude": -122.128451  
                },  
                "btmRightPoint": {  
                    "latitude": 47.630499,  
                    "longitude": -122.12582  
                }  
            }  
        },  
        {  
            "id": "522---2060604",  
            "type": "stop",  
            "objectDetails": {  
                "stopKey": "71310",  
                "stopName": "NE 24th St & 160th Ave NE",  
                "stopCode": "71310",  
                "mainTransitType": "Bus",  
                "mainAgencyId": "522---5872",  
                "mainAgencyName": "Metro Transit"  
            },  
            "position": {  
                "latitude": 47.631409,  
                "longitude": -122.127136  
            },  
            "viewport": {  
                "topLeftPoint": {  
                    "latitude": 47.632318,  
                    "longitude": -122.128451  
                },  
                "btmRightPoint": {  
                    "latitude": 47.630499,  
                    "longitude": -122.12582  
                }  
            }  
        }  
    ]  
}
```

```

        "stopName": "NE 24th St & 160th Ave NE",
        "stopCode": "71310",
        "mainTransitType": "Bus",
        "mainAgencyId": "522---5872",
        "mainAgencyName": "Metro Transit"
    },
    "position": {
        "latitude": 47.631565,
        "longitude": -122.127808
    },
    "viewport": {
        "topLeftPoint": {
            "latitude": 47.632474,
            "longitude": -122.129124
        },
        "bottomRightPoint": {
            "latitude": 47.630655,
            "longitude": -122.126492
        }
    }
}
]
}

```

If you observe the response structure carefully, you'll see that it contains parameters for each transit object. Each transit object has parameters such as `id`, `type`, `stopName`, `mainTransitType`, `mainAgencyName`, and the position, in coordinates, of the object.

For the purpose of learning, we'll use an `id` of a bus stops as the origin, for our route in the next section.

Request a transit route

The Azure Maps [Get Transit Routes API](#) allows trip planning. It returns the best possible route options from an origin to a destination. The service provides different kind of travel modes, including walking, biking, and public transit. Next, we'll search a route from the closest bus stop to the Space Needle tower in Seattle.

Get location coordinates for destination

To obtain the location coordinates of the Space Needle tower, we'll use the Azure Maps [Fuzzy Search service](#).

To make a request to the Fuzzy search service, follow the steps below:

1. In Postman, click **New Request | GET request** and name it **Get location coordinates**.
2. On the Builder tab, select the **GET** HTTP method, enter the following request URL, and click **Send**.

```
https://atlas.microsoft.com/search/fuzzy/json?subscription-key={subscription-key}&api-version=1.0&query=space needle
```

3. If you look at the response carefully, it contains multiple locations in the results for the Space Needle search. Each result contains the location coordinates under the **position**. Copy the `lat` and `lon` under the **position** of the first result.

```
{
    "summary": {
        "query": "space needle",
        "queryType": "NON_NEAR",
        "queryTime": 35,
        "numResults": 8,
        "offset": 0,
        "totalResults": 11,
        "fuzzyLevel": 1
    }
}
```

```
},
"results": [
{
    "type": "POI",
    "id": "US/POI/p0/6993440",
    "score": 4.67369,
    "info": "search:ta:840539001406144-US",
    "poi": {
        "name": "Space Needle",
        "phone": "+(1)-(206)-9052100",
        "categorySet": [
            {
                "id": 7376009
            }
        ],
        "url": "www.spaceneedle.com",
        "categories": [
            "important tourist attraction",
            "tower"
        ],
        "classifications": [
            {
                "code": "IMPORTANT_TOURIST_ATTRACTION",
                "names": [
                    {
                        "nameLocale": "en-US",
                        "name": "important tourist attraction"
                    },
                    {
                        "nameLocale": "en-US",
                        "name": "tower"
                    }
                ]
            }
        ],
        "address": {
            "streetNumber": "400",
            "streetName": "Broad St",
            "municipalitySubdivision": "South Lake Union, Seattle, Lower Queen Anne",
            "municipality": "Seattle",
            "countrySecondarySubdivision": "King",
            "countryTertiarySubdivision": "Seattle",
            "countrySubdivision": "WA",
            "postalCode": "98109",
            "countryCode": "US",
            "country": "United States",
            "countryCodeISO3": "USA",
            "freeformAddress": "400 Broad St, Seattle, WA 98109",
            "localName": "Seattle",
            "countrySubdivisionName": "Washington"
        },
        "position": {
            "lat": 47.62039,
            "lon": -122.34928
        },
        "viewport": {
            "topLeftPoint": {
                "lat": 47.62129,
                "lon": -122.35061
            },
            "bottomRightPoint": {
                "lat": 47.61949,
                "lon": -122.34795
            }
        },
        "entryPoints": [
            {
                "type": "main",
                "id": 7376009
            }
        ]
    }
}];
```

```

        "position": {
            "lat": 47.61982,
            "lon": -122.34886
        }
    },
    ...
    ...
    ...
]
}

```

Request route

To make a route request, complete the steps below:

1. In Postman, click **New Request | GET request** and name it **Get Route info**.
2. On the Builder tab, select the **GET** HTTP method, enter the following request URL for your API endpoint and click **Send**.

We'll request public transit routes for a bus by specifying the `modeType` and `transitType` parameters. The request URL contains the locations retrieved in the previous sections. For the `originType`, we now have a `stopId`. And for the `destinationType`, we have the `position`.

See the [list of URI parameters](#) you can use in your request to the [Get Transit Routes API](#).

```
https://atlas.microsoft.com/mobility/transit/route/json?subscription-key={subscription-key}&api-version=1.0&originType=stopId&origin=522---2060603&destinationType=position&destination=47.62039,-122.34928&modeType=publicTransit&transitType=bus
```

3. Upon a successful request, the response structure should look like the one below:

```
{
  "results": [
    {
      "itineraryId": "cb6b6b6f-5cda-451e-b68d-2e97971dd60c---20190906BBBEC4D2219A436A9D79424978C9BBF:0---522",
      "departureTime": "2019-09-07T01:01:50Z",
      "arrivalTime": "2019-09-07T02:16:33Z",
      "travelTimeInSeconds": 4483,
      "numberOfLegs": 8,
      "legs": [
        {
          "legType": "Wait",
          "legStartTime": "2019-09-07T01:01:50Z",
          "legEndTime": "2019-09-07T01:01:50Z",
          "caption": "249"
        },
        {
          "legType": "Bus",
          "legStartTime": "2019-09-07T01:01:50Z",
          "legEndTime": "2019-09-07T01:26:00Z",
          "caption": "249",
          "lengthInMeters": 9139
        },
        {
          "legType": "Wait",
          "legStartTime": "2019-09-07T01:26:00Z",
          "legEndTime": "2019-09-07T01:28:00Z",
          "caption": "255"
        },
        {
          "legType": "Walk"
        }
      ]
    }
  ]
}
```

```
        "legType": "Bus",
        "legStartTime": "2019-09-07T01:28:00Z",
        "legEndTime": "2019-09-07T01:57:21Z",
        "caption": "255",
        "lengthInMeters": 13136
    },
    {
        "legType": "Walk",
        "legStartTime": "2019-09-07T01:57:22Z",
        "legEndTime": "2019-09-07T02:01:27Z",
        "caption": "Denny Way",
        "lengthInMeters": 308
    },
    {
        "legType": "Wait",
        "legStartTime": "2019-09-07T02:01:27Z",
        "legEndTime": "2019-09-07T02:06:33Z",
        "caption": "8"
    },
    {
        "legType": "Bus",
        "legStartTime": "2019-09-07T02:06:33Z",
        "legEndTime": "2019-09-07T02:12:41Z",
        "caption": "8",
        "lengthInMeters": 1060
    },
    {
        "legType": "Walk",
        "legStartTime": "2019-09-07T02:12:42Z",
        "legEndTime": "2019-09-07T02:16:33Z",
        "lengthInMeters": 251
    }
],
"itineraryFare": {
    "price": {
        "amount": 550,
        "currencyCode": "USD"
    },
    "tickets": [
        {
            "amount": 275,
            "currencyCode": "USD"
        },
        {
            "amount": 275,
            "currencyCode": "USD"
        }
    ]
},
...
{
    "itineraryId": "cb6b6b6f-5cda-451e-b68d-2e97971dd60c---20190906BBBEC4D2219A436A9D794224978C9BBF:2---522",
    "departureTime": "2019-09-07T00:49:32Z",
    "arrivalTime": "2019-09-07T02:20:06Z",
    "travelTimeInSeconds": 5434,
    "numberOfLegs": 10,
    "legs": [
        {
            "legType": "Wait",
            "legStartTime": "2019-09-07T00:49:32Z",
            "legEndTime": "2019-09-07T00:49:32Z",
            "caption": "226"
        },
        {
            "legType": "Bus",
            "legStartTime": "2019-09-07T00:49:32Z",
            "legEndTime": "2019-09-07T01:15:00Z",
            "caption": "227"
        }
    ]
}
```

```
        "caption": "226",
        "lengthInMeters": 6792
    },
    {
        "legType": "Wait",
        "legStartTime": "2019-09-07T01:15:00Z",
        "legEndTime": "2019-09-07T01:20:00Z",
        "caption": "241"
    },
    {
        "legType": "Bus",
        "legStartTime": "2019-09-07T01:20:00Z",
        "legEndTime": "2019-09-07T01:28:00Z",
        "caption": "241",
        "lengthInMeters": 3397
    },
    {
        "legType": "Wait",
        "legStartTime": "2019-09-07T01:28:00Z",
        "legEndTime": "2019-09-07T01:33:00Z",
        "caption": "550"
    },
    {
        "legType": "Bus",
        "legStartTime": "2019-09-07T01:33:00Z",
        "legEndTime": "2019-09-07T01:58:00Z",
        "caption": "550",
        "lengthInMeters": 12899
    },
    {
        "legType": "Walk",
        "legStartTime": "2019-09-07T01:58:01Z",
        "legEndTime": "2019-09-07T01:59:21Z",
        "caption": "4th Avenue South",
        "lengthInMeters": 99
    },
    {
        "legType": "Wait",
        "legStartTime": "2019-09-07T01:59:21Z",
        "legEndTime": "2019-09-07T02:01:00Z",
        "caption": "33"
    },
    {
        "legType": "Bus",
        "legStartTime": "2019-09-07T02:01:00Z",
        "legEndTime": "2019-09-07T02:13:29Z",
        "caption": "33,24",
        "lengthInMeters": 2447
    },
    {
        "legType": "Walk",
        "legStartTime": "2019-09-07T02:13:30Z",
        "legEndTime": "2019-09-07T02:20:06Z",
        "lengthInMeters": 457
    }
],
"itineraryFare": {
    "price": {
        "amount": 550,
        "currencyCode": "USD"
    },
    "tickets": [
        {
            "amount": 275,
            "currencyCode": "USD"
        },
        {
            "amount": 275,
            "currencyCode": "USD"
        }
    ]
}
```

```
        }
    ]
}
]
```

4. If you observe carefully, there are multiple **bus** routes in the response. Each route has a unique **itinerary ID**, a summary that describes each leg of the route, and an **itineraryFare** that gives both the itemized and total price for bus tickets. A route leg is the part of the route between two stop waypoints. Next, we'll request details for the fastest route using the **itineraryId** in the response.

Request fastest route itinerary

The Azure Maps [Get Transit Itinerary](#) service allows you to request data for a particular route using the route's **itinerary ID** returned by the [Get Transit Routes API](#) service. To make a request, complete the steps below:

1. In Postman, click **New Request | GET request** and name it **Get Transit info**.
2. On the Builder tab, select the **GET** HTTP method. Enter the following request URL for your API endpoint and click **Send**.

We'll set the **detailType** parameter to **geometry** so that the response contains stop information for public transit and turn-by-turn navigation for walk and bike legs of the route.

```
https://atlas.microsoft.com/mobility/transit/itinerary/json?api-version=1.0&subscription-key={subscription-key}&query={itineraryId}&detailType=geometry
```

3. Upon a successful request, the response structure should look like the one below. If you observe the JSON response, you'll notice that each bus leg contains a **legfare** element. The **legfare** element contains the cost, in cents, of each bus route that is purchased separately. At the end of the response, you'll see an **itineraryFare** element that contains the cost, in cents, of the entire route. In this example, there are four bus routes at **\$2.75** each. However, if you purchase a single ticket for the entire route, the cost is **\$5.50**.

```
{
  "departureTime": "2020-07-22T19:54:47Z",
  "arrivalTime": "2020-07-22T21:12:21Z",
  "legs": [
    {
      "legType": "Wait",
      "legStartTime": "2020-07-22T19:54:47Z",
      "legEndTime": "2020-07-22T19:54:47Z",
      "lineGroup": {
        "lineGroupId": "522---666063",
        "agencyId": "522---5872",
        "agencyName": "Metro Transit",
        "lineNumber": "226",
        "caption1": "Eastgate P&R-Crossroads-Overlake-Bellevue TC",
        "caption2": "226 Eastgate P&R-Crossroads-Overlake-Bellevue TC",
        "color": "347E5D",
        "transitType": "Bus"
      },
      "line": {
        "lineId": "522---2756599",
        "lineGroupId": "522---666063",
        "direction": "forward",
        "agencyId": "522---5872",
        "lineNumber": "226",
        "lineDestination": "Bellevue Transit Center Crossroads"
      }
    }
  ]
}
```

```
},
"stops": [
{
    "stopId": "522---2060603",
    "stopKey": "71300",
    "stopName": "NE 24th St & 162nd Ave NE",
    "stopCode": "71300",
    "position": {
        "latitude": 47.631504,
        "longitude": -122.125275
    },
    "mainTransitType": "Bus",
    "mainAgencyId": "522---5872",
    "mainAgencyName": "Metro Transit"
},
{
    "stopId": "522---2062263",
    "stopKey": "85630",
    "stopName": "Bellevue Tc",
    "stopCode": "85630",
    "position": {
        "latitude": 47.615591,
        "longitude": -122.196491
    },
    "mainTransitType": "Bus",
    "mainAgencyId": "522---5872",
    "mainAgencyName": "Metro Transit"
}
],
"waitOnVehicle": false
},
{
    "legType": "Bus",
    "legStartTime": "2020-07-22T19:54:47Z",
    "legEndTime": "2020-07-22T20:15:00Z",
    "lineGroup": {
        "lineGroupId": "522---666063",
        "agencyId": "522---5872",
        "agencyName": "Metro Transit",
        "lineNumber": "226",
        "caption1": "Eastgate P&R-Crossroads-Overlake-Bellevue TC",
        "caption2": "226 Eastgate P&R-Crossroads-Overlake-Bellevue TC",
        "color": "347E5D",
        "transitType": "Bus"
    },
    "line": {
        "lineId": "522---2756599",
        "lineGroupId": "522---666063",
        "direction": "forward",
        "agencyId": "522---5872",
        "lineNumber": "226",
        "lineDestination": "Bellevue Transit Center Crossroads"
    },
    "stops": [
        ...
    ],
    "geometry": {
        "type": "LineString",
        "coordinates": [
            ...
        ]
    },
    "legFare": {
        "fares": [
            {
                "price": {
                    "amount": 275,
                    "currencyCode": "USD"
                }
            }
        ]
    }
}
```

```
        "usage": "pay"
    }
]
}
},
...
...
{
    "legType": "Bus",
    "legStartTime": "2020-07-22T20:20:00Z",
    "legEndTime": "2020-07-22T20:28:00Z",
    "lineGroup": {
        "lineGroupId": "522---666071",
        "agencyId": "522---5872",
        "agencyName": "Metro Transit",
        "lineNumber": "241",
        "caption1": "Eastgate P&R - Bellevue Transit Center",
        "caption2": "241 Eastgate P&R - Bellevue Transit Center",
        "color": "347E5D",
        "transitType": "Bus"
    },
    "line": {
        "lineId": "522---2756619",
        "lineGroupId": "522---666071",
        "direction": "backward",
        "agencyId": "522---5872",
        "lineNumber": "241",
        "lineDestination": "Eastgate P&R Factoria"
    },
    "stops": [
        ...
    ],
    "geometry": {
        "type": "LineString",
        "coordinates": [
            ...
        ]
    },
    "legFare": {
        "fares": [
            {
                "price": {
                    "amount": 275,
                    "currencyCode": "USD"
                },
                "usage": "transfer"
            }
        ]
    }
},
...
{
    "legType": "Bus",
    "legStartTime": "2020-07-22T20:31:00Z",
    "legEndTime": "2020-07-22T20:54:13Z",
    "lineGroup": {
        "lineGroupId": "522---312636",
        "agencyId": "522---854535",
        "agencyName": "Sound Transit",
        "lineNumber": "550",
        "caption1": "Bellevue - Seattle",
        "caption2": "550 Bellevue - Seattle",
        "color": "00008B",
        "transitType": "Bus"
    },
    "line": {
        "lineId": "522---962201",
        "lineGroupId": "522---312636",
        "direction": "backward",
        "agencyId": "522---854535",
        "lineNumber": "550",
        "lineDestination": "Seattle - Bellevue"
    }
}
```

```
        "agencyId": "522---854535",
        "lineNumber": "550",
        "lineDestination": "Seattle"
    },
    "stops": [
        ...
    ],
    "geometry": {
        "type": "LineString",
        "coordinates": [
            ...
        ]
    },
    "legFare": {
        "fares": [
            {
                "price": {
                    "amount": 275,
                    "currencyCode": "USD"
                },
                "usage": "pay"
            }
        ]
    }
},
...
...
{
    "legType": "Bus",
    "legStartTime": "2020-07-22T20:57:00Z",
    "legEndTime": "2020-07-22T21:06:00Z",
    "lineGroup": {
        "lineGroupId": "522---480518",
        "agencyId": "522---5872",
        "agencyName": "Metro Transit",
        "lineNumber": "13",
        "caption1": "Seattle Pacific - Downtown Seattle",
        "caption2": "13 Seattle Pacific - Downtown Seattle",
        "color": "347E5D",
        "transitType": "Bus"
    },
    "line": {
        "lineId": "522---1744932",
        "lineGroupId": "522---480518",
        "direction": "forward",
        "agencyId": "522---5872",
        "lineNumber": "13",
        "lineDestination": "Seattle Pacific University Seattle Center W"
    },
    "stops": [
        ...
    ],
    "geometry": {
        "type": "LineString",
        "coordinates": [
            ...
        ]
    },
    "legFare": {
        "fares": [
            {
                "price": {
                    "amount": 275,
                    "currencyCode": "USD"
                },
                "usage": "transfer"
            }
        ]
    }
}
```

```
        },
        ...
    ],
    "itineraryFare": {
        "price": {
            "amount": 550,
            "currencyCode": "USD"
        },
        "tickets": [
            {
                "amount": 275,
                "currencyCode": "USD"
            },
            {
                "amount": 275,
                "currencyCode": "USD"
            }
        ]
    }
}
```

Next steps

Learn how to request real-time data using Mobility services (Preview):

[How to request real-time data](#)

Explore the Azure Maps Mobility services (Preview) API documentation

[Mobility services documentation](#)

Request real-time public transit data using the Azure Maps Mobility services (Preview)

3/5/2021 • 2 minutes to read • [Edit Online](#)

IMPORTANT

Azure Maps Mobility services are currently in public preview. This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

This article shows you how to use Azure Maps [Mobility services](#) to request real-time public transit data.

In this article, you will learn how to request next real-time arrivals for all lines arriving at a given stop

Prerequisites

You first need to have an Azure Maps account and a subscription key to make any calls to the Azure Maps public transit APIs. For information, follow instructions in [Create an account](#) to create an Azure Maps account. Follow the steps in [get primary key](#) to obtain the primary key for your account. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

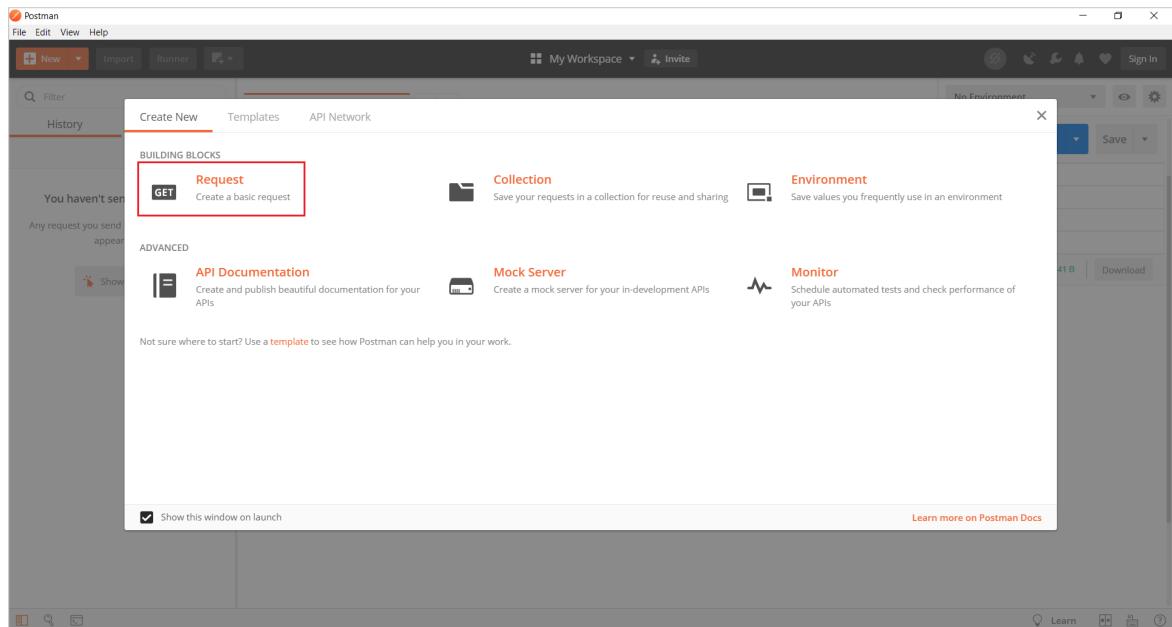
This article uses the [Postman app](#) to build REST calls. You can use any API development environment that you prefer.

Request real-time arrivals for a stop

In order to request real-time arrivals data of a particular public transit stop, you'll need to make request to the [Real-time Arrivals API](#) of the Azure Maps [Mobility Service \(Preview\)](#). You'll need the **metroID** and **stopID** to complete the request. To learn more about how to request these parameters, see our guide on how to [request public transit routes](#).

Let's use "522" as our metro ID, which is the metro ID for the "Seattle–Tacoma–Bellevue, WA" area. Use "522---2060603" as the stop ID, this bus stop is at "Ne 24th St & 162nd Ave Ne, Bellevue WA". To request the next five real-time arrivals data, for all next live arrivals at this stop, complete the following steps:

1. Open the Postman app, and let's create a collection to store the requests. Near the top of the Postman app, select **New**. In the **Create New** window, select **Collection**. Name the collection and select the **Create** button.
2. To create the request, select **New** again. In the **Create New** window, select **Request**. Enter a **Request name** for the request. Select the collection you created in the previous step, as the location in which to save the request. Then, select **Save**.



3. Select the **GET** HTTP method on the builder tab and enter the following URL to create a GET request.
Replace `{subscription-key}`, with your Azure Maps primary key.

```
https://atlas.microsoft.com/mobility/realtme/arrivals/json?subscription-key={subscription-key}&api-version=1.0&metroId=522&query=522---2060603&transitType=bus
```

4. After a successful request, you'll receive the following response. Notice that parameter 'scheduleType' defines whether the estimated arrival time is based on real-time or static data.

```

{
  "results": [
    {
      "arrivalMinutes": 8,
      "scheduleType": "realTime",
      "patternId": "522---4143196",
      "line": {
        "lineId": "522---3760143",
        "lineGroupId": "522---666077",
        "direction": "backward",
        "agencyId": "522---5872",
        "agencyName": "Metro Transit",
        "lineNumber": "249",
        "lineDestination": "South Bellevue S Kirkland P&R",
        "transitType": "Bus"
      },
      "stop": {
        "stopId": "522---2060603",
        "stopKey": "71300",
        "stopName": "NE 24th St & 162nd Ave NE",
        "stopCode": "71300",
        "position": {
          "latitude": 47.631504,
          "longitude": -122.125275
        },
        "mainTransitType": "Bus",
        "mainAgencyId": "522---5872",
        "mainAgencyName": "Metro Transit"
      }
    },
    {
      "arrivalMinutes": 25,
      "scheduleType": "realTime",
      "patternId": "522---3510227",
      "line": {
        "lineId": "522---2756599",
        "lineGroupId": "522---666063",
        "direction": "forward",
        "agencyId": "522---5872",
        "agencyName": "Metro Transit",
        "lineNumber": "226",
        "lineDestination": "Bellevue Transit Center Crossroads",
        "transitType": "Bus"
      },
      "stop": {
        "stopId": "522---2060603",
        "stopKey": "71300",
        "stopName": "NE 24th St & 162nd Ave NE",
        "stopCode": "71300",
        "position": {
          "latitude": 47.631504,
          "longitude": -122.125275
        },
        "mainTransitType": "Bus",
        "mainAgencyId": "522---5872",
        "mainAgencyName": "Metro Transit"
      }
    }
  ]
}

```

Next steps

Learn how to request transit data using Mobility services (Preview):

[How to request transit data](#)

Explore the Azure Maps Mobility services (Preview) API documentation:

[Mobility services API documentation](#)

Request real-time and forecasted weather data using Azure Maps Weather services

4/30/2021 • 12 minutes to read • [Edit Online](#)

Azure Maps [Weather services](#) are a set of RESTful APIs that allows developers to integrate highly dynamic historical, real-time, and forecasted weather data and visualizations into their solutions. In this article, we'll show you how to request both real-time and forecasted weather data.

In this article you'll learn, how to:

- Request real-time (current) weather data using the [Get Current Conditions API](#).
- Request severe weather alerts using the [Get Severe Weather Alerts API](#).
- Request daily forecasts using the [Get Daily Forecast API](#).
- Request hourly forecasts using the [Get Hourly Forecast API](#).
- Request minute by minute forecasts using the [Get Minute Forecast API](#).

This video provides examples for making REST calls to Azure Maps Weather services.

Prerequisites

1. [Make an Azure Maps account](#)
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key. For more information on authentication in Azure Maps, see [manage authentication in Azure Maps](#).

IMPORTANT

The [Get Minute Forecast API](#) requires a Gen 1 (S1) or Gen 2 pricing tier. All other APIs require an S0 pricing tier key.

This tutorial uses the [Postman](#) application, but you may choose a different API development environment.

Request real-time weather data

The [Get Current Conditions API](#) returns detailed weather conditions such as precipitation, temperature, and wind for a given coordinate location. Also, observations from the past 6 or 24 hours for a particular location can be retrieved. The response includes details like observation date and time, brief description of the weather conditions, weather icon, precipitation indicator flags, and temperature. RealFeel™ Temperature and ultraviolet(UV) index are also returned.

In this example, you'll use the [Get Current Conditions API](#) to retrieve current weather conditions at coordinates located in Seattle, WA.

1. Open the Postman app. Near the top of the Postman app, select **New**. In the **Create New** window, select **Collection**. Name the collection and select the **Create** button. You'll use this collection for the rest of the examples in this document.

2. To create the request, select **New** again. In the **Create New** window, select **Request**. Enter a **Request name** for the request. Select the collection you created in the previous step, and then select **Save**.

3. Select the **GET** HTTP method in the builder tab and enter the following URL. For this request, and other requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key.

```
https://atlas.microsoft.com/weather/currentConditions/json?api-version=1.0&query=47.60357,-122.32945&subscription-key={Azure-Maps-Primary-Subscription-key}
```

4. Click the blue **Send** button. The response body contains current weather information.

```
{
  "results": [
    {
      "dateTime": "2020-10-19T20:39:00+00:00",
      "phrase": "Cloudy",
      "iconCode": 7,
      "hasPrecipitation": false,
      "isDayTime": true,
      "temperature": {
        "value": 12.4,
        "unit": "C",
        "unitType": 17
      },
      "realFeelTemperature": {
        "value": 13.7,
        "unit": "C",
        "unitType": 17
      },
      "realFeelTemperatureShade": {
        "value": 13.7,
        "unit": "C",
        "unitType": 17
      },
      "relativeHumidity": 87,
      "dewPoint": {
        "value": 10.3,
        "unit": "C",
        "unitType": 17
      },
      "wind": {
        "direction": {
          "degrees": 23.0,
          "localizedDescription": "NNE"
        },
        "speed": {
          "value": 4.5,
          "unit": "km/h",
          "unitType": 7
        }
      },
      "windGust": {
        "speed": {
          "value": 9.0,
          "unit": "km/h",
          "unitType": 7
        }
      },
      "uvIndex": 1,
      "uvIndexPhrase": "Low",
      "visibility": {
        "value": 9.7,
        "unit": "km",
        "unitType": 6
      }
    }
  ]
}
```

```
},
"obstructionsToVisibility": "",
"cloudCover": 100,
"ceiling": {
    "value": 1494.0,
    "unit": "m",
    "unitType": 5
},
"pressure": {
    "value": 1021.2,
    "unit": "mb",
    "unitType": 14
},
"pressureTendency": {
    "localizedDescription": "Steady",
    "code": "S"
},
"past24HourTemperatureDeparture": {
    "value": -2.1,
    "unit": "C",
    "unitType": 17
},
"apparentTemperature": {
    "value": 15.0,
    "unit": "C",
    "unitType": 17
},
"windChillTemperature": {
    "value": 12.2,
    "unit": "C",
    "unitType": 17
},
"wetBulbTemperature": {
    "value": 11.3,
    "unit": "C",
    "unitType": 17
},
"precipitationSummary": {
    "pastHour": {
        "value": 0.0,
        "unit": "mm",
        "unitType": 3
    },
    "past3Hours": {
        "value": 0.0,
        "unit": "mm",
        "unitType": 3
    },
    "past6Hours": {
        "value": 0.0,
        "unit": "mm",
        "unitType": 3
    },
    "past9Hours": {
        "value": 0.0,
        "unit": "mm",
        "unitType": 3
    },
    "past12Hours": {
        "value": 0.0,
        "unit": "mm",
        "unitType": 3
    },
    "past18Hours": {
        "value": 0.0,
        "unit": "mm",
        "unitType": 3
    },
    "past24Hours": {
```

```
        "value": 0.4,
        "unit": "mm",
        "unitType": 3
    }
},
"temperatureSummary": {
    "past6Hours": {
        "minimum": {
            "value": 12.2,
            "unit": "C",
            "unitType": 17
        },
        "maximum": {
            "value": 14.0,
            "unit": "C",
            "unitType": 17
        }
    },
    "past12Hours": {
        "minimum": {
            "value": 12.2,
            "unit": "C",
            "unitType": 17
        },
        "maximum": {
            "value": 14.0,
            "unit": "C",
            "unitType": 17
        }
    },
    "past24Hours": {
        "minimum": {
            "value": 12.2,
            "unit": "C",
            "unitType": 17
        },
        "maximum": {
            "value": 15.6,
            "unit": "C",
            "unitType": 17
        }
    }
}
]
}
```

Request severe weather alerts

Azure Maps Get Severe Weather Alerts API returns the severeweatheralerts that are available worldwide from both official Government Meteorological Agencies and leading globalto regional weatheralert providers. The service can return details such as alert type, category, level, and detailed descriptions about the active severe alerts for the requested location, such as hurricanes, thunderstorms, lightning, heat waves or forest fires. As an example, logistics managers can visualize severe weather conditions on a map, along with business locations and planned routes, and coordinate further with drivers and local workers.

In this example, you'll use the [Get Severe Weather Alerts API](#) to retrieve current weather conditions at coordinates located in Cheyenne, WY.

NOTE

This example retrieves severe weather alerts at the time of this writing. It is likely that there are no longer any severe weather alerts at the requested location. To retrieve actual severe alert data when running this example, you'll need to retrieve data at a different coordinate location.

1. Open the Postman app, click **New**, and select **Request**. Enter a **Request name** for the request. Select the collection you created in the previous section or created a new one, and then select **Save**.
2. Select the **GET** HTTP method in the builder tab and enter the following URL. For this request, and other requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key.

```
https://atlas.microsoft.com/weather/severe/alerts/json?api-version=1.0&query=41.161079,-104.805450&subscription-key={Azure-Maps-Primary-Subscription-key}
```

3. Click the blue **Send** button. If there are no severe weather alerts, the response body will contain an empty `results[]` array. If there are severe weather alerts, the response body contains something like the following JSON response:

```
{
  "results": [
    {
      "countryCode": "US",
      "alertId": 2194734,
      "description": {
        "localized": "Red Flag Warning",
        "english": "Red Flag Warning"
      },
      "category": "FIRE",
      "priority": 54,
      "source": "U.S. National Weather Service",
      "sourceId": 2,
      "alertAreas": [
        {
          "name": "Platte/Goshen/Central and Eastern Laramie",
          "summary": "Red Flag Warning in effect until 7:00 PM MDT. Source: U.S. National Weather Service",
          "startTime": "2020-10-05T15:00:00+00:00",
          "endTime": "2020-10-06T01:00:00+00:00",
          "latestStatus": {
            "localized": "Continue",
            "english": "Continue"
          },
          "alertDetails": "...RED FLAG WARNING REMAINS IN EFFECT FROM 9 AM THIS MORNING TO\n7 PM MDT THIS EVENING FOR STRONG GUSTY WINDS AND LOW HUMIDITY...\n* WHERE...Fire weather zones 303, 304, 305, 306, 307, 308, 309, and 310 in southeast Wyoming. Fire weather zone 313 in Nebraska.\n* WIND...West to northwest 15 to 30 MPH with gusts around 40 MPH.\n* HUMIDITY...10 to 15 percent.\n* IMPACTS...Any fires that develop will likely spread rapidly.\n Outdoor burning is not recommended.\nPRECAUTIONARY/PREPAREDNESS ACTIONS...\nA Red Flag Warning means that critical fire weather conditions\nare either occurring now...or will shortly. A combination of\nstrong winds...low relative humidity...and warm temperatures can\ncontribute to extreme fire behavior.\n",
          "alertDetailsLanguageCode": "en"
        }
      ]
    },...
  ]
}
```

Request daily weather forecast data

The [Get Daily Forecast API](#) returns detailed daily weather forecast such as temperature and wind. The request can specify how many days to return: 1, 5, 10, 15, 25, or 45 days for a given coordinate location. The response includes details such as temperature, wind, precipitation, air quality, and UV index. In this example, we request for five days by setting `duration=5`.

IMPORTANT

In the S0 pricing tier, you can request daily forecast for the next 1, 5, 10, and 15 days. In either Gen 1 (S1) or Gen 2 pricing tier, you can request daily forecast for the next 25 days, and 45 days.

In this example, you'll use the [Get Daily Forecast API](#) to retrieve the five-day weather forecast for coordinates located in Seattle, WA.

1. Open the Postman app, click **New**, and select **Request**. Enter a **Request name** for the request. Select the collection you created in the previous section or created a new one, and then select **Save**.
2. Select the **GET** HTTP method in the builder tab and enter the following URL. For this request, and other requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key.

```
https://atlas.microsoft.com/weather/forecast/daily/json?api-version=1.0&query=47.60357,-122.32945&duration=5&subscription-key={Azure-Maps-Primary-Subscription-key}
```

3. Click the blue **Send** button. The response body contains the five-day weather forecast data. For the sake of brevity, the JSON response below shows the forecast for the first day.

```
{
  "summary": {
    "startDate": "2020-10-18T17:00:00+00:00",
    "endDate": "2020-10-19T23:00:00+00:00",
    "severity": 2,
    "phrase": "Snow, mixed with rain at times continuing through Monday evening and a storm total of 3-6 cm",
    "category": "snow/rain"
  },
  "forecasts": [
    {
      "date": "2020-10-19T04:00:00+00:00",
      "temperature": {
        "minimum": {
          "value": -1.1,
          "unit": "C",
          "unitType": 17
        },
        "maximum": {
          "value": 1.3,
          "unit": "C",
          "unitType": 17
        }
      },
      "realFeelTemperature": {
        "minimum": {
          "value": -6.0,
          "unit": "C",
          "unitType": 17
        },
        "maximum": {
          "value": 0.5,
          "unit": "C"
        }
      }
    }
  ]
}
```

```
        "unit": "C",
        "unitType": 17
    },
    "realFeelTemperatureShade": {
        "minimum": {
            "value": -6.0,
            "unit": "C",
            "unitType": 17
        },
        "maximum": {
            "value": 0.7,
            "unit": "C",
            "unitType": 17
        }
    },
    "hoursOfSun": 1.8,
    "degreeDaySummary": {
        "heating": {
            "value": 18.0,
            "unit": "C",
            "unitType": 17
        },
        "cooling": {
            "value": 0.0,
            "unit": "C",
            "unitType": 17
        }
    },
    "airAndPollen": [
        {
            "name": "AirQuality",
            "value": 23,
            "category": "Good",
            "categoryValue": 1,
            "type": "Ozone"
        },
        {
            "name": "Grass",
            "value": 0,
            "category": "Low",
            "categoryValue": 1
        },
        {
            "name": "Mold",
            "value": 0,
            "category": "Low",
            "categoryValue": 1
        },
        {
            "name": "Ragweed",
            "value": 0,
            "category": "Low",
            "categoryValue": 1
        },
        {
            "name": "Tree",
            "value": 0,
            "category": "Low",
            "categoryValue": 1
        },
        {
            "name": "UVIndex",
            "value": 0,
            "category": "Low",
            "categoryValue": 1
        }
    ],
    "day": {
        "sunrise": "2023-09-22T06:15:00Z",
        "sunset": "2023-09-22T18:45:00Z",
        "dawn": "2023-09-22T05:00:00Z",
        "dusk": "2023-09-22T20:00:00Z",
        "twilight": "2023-09-22T06:00:00Z",
        "night": "2023-09-22T19:00:00Z"
    }
]
```

```
"iconCode": 22,
"iconPhrase": "Snow",
"hasPrecipitation": true,
"precipitationType": "Mixed",
"precipitationIntensity": "Light",
"shortPhrase": "Chilly with snow, 2-4 cm",
"longPhrase": "Chilly with snow, accumulating an additional 2-4 cm",
"precipitationProbability": 90,

```

```

    "precipitationProbability": 65,
    "thunderstormProbability": 0,
    "rainProbability": 60,
    "snowProbability": 54,
    "iceProbability": 4,
    "wind": {
        "direction": {
            "degrees": 16.0,
            "localizedDescription": "NNE"
        },
        "speed": {
            "value": 16.7,
            "unit": "km/h",
            "unitType": 7
        }
    },
    "windGust": {
        "direction": {
            "degrees": 1.0,
            "localizedDescription": "N"
        },
        "speed": {
            "value": 35.2,
            "unit": "km/h",
            "unitType": 7
        }
    },
    "totalLiquid": {
        "value": 4.3,
        "unit": "mm",
        "unitType": 3
    },
    "rain": {
        "value": 3.0,
        "unit": "mm",
        "unitType": 3
    },
    "snow": {
        "value": 0.79,
        "unit": "cm",
        "unitType": 4
    },
    "ice": {
        "value": 0.0,
        "unit": "mm",
        "unitType": 3
    },
    "hoursOfPrecipitation": 4.0,
    "hoursOfRain": 1.0,
    "hoursOfSnow": 3.0,
    "hoursOfIce": 0.0,
    "cloudCover": 94
},
"sources": [
    "AccuWeather"
]
},
...
]
}

```

Request hourly weather forecast data

The [Get Hourly Forecast API](#) returns detailed weather forecast by the hour for the next 1, 12, 24 (1 day), 72 (3 days), 120 (5 days), and 240 hours (10 days) for the given coordinate location. The API returns details such as temperature, humidity, wind, precipitation, and UV index.

IMPORTANT

In the S0 pricing tier, you can request hourly forecast for the next 1, 12, 24 hours (1 day), and 72 hours (3 days). In either Gen 1 (S1) or Gen 2 pricing tier, you can request hourly forecast for the next 120 (5 days) and 240 hours (10 days).

In this example, you'll use the [Get Hourly Forecast API](#) to retrieve the hourly weather forecast for the next 12 hours at coordinates located in Seattle, WA.

1. Open the Postman app, click **New**, and select **Request**. Enter a **Request name** for the request. Select the collection you created in the previous section or created a new one, and then select **Save**.
2. Select the **GET** HTTP method in the builder tab and enter the following URL. For this request, and other requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key.

```
https://atlas.microsoft.com/weather/forecast/hourly/json?api-version=1.0&query=47.60357,-122.32945&duration=12&subscription-key={Azure-Maps-Primary-Subscription-key}
```

3. Click the blue **Send** button. The response body contains weather forecast data for the next 12 hours. For the sake of brevity, the JSON response below shows the forecast for the first hour.

```
{
  "forecasts": [
    {
      "date": "2020-10-19T21:00:00+00:00",
      "iconCode": 12,
      "iconPhrase": "Showers",
      "hasPrecipitation": true,
      "precipitationType": "Rain",
      "precipitationIntensity": "Light",
      "isDaylight": true,
      "temperature": {
        "value": 14.7,
        "unit": "C",
        "unitType": 17
      },
      "realFeelTemperature": {
        "value": 13.3,
        "unit": "C",
        "unitType": 17
      },
      "wetBulbTemperature": {
        "value": 12.0,
        "unit": "C",
        "unitType": 17
      },
      "dewPoint": {
        "value": 9.5,
        "unit": "C",
        "unitType": 17
      },
      "wind": {
        "direction": {
          "degrees": 242.0,
          "localizedDescription": "WSW"
        },
        "speed": {
          "value": 9.3,
          "unit": "km/h",
          "unitType": 7
        }
      }
    }
  ]
}
```

```

        "windGust": {
            "speed": {
                "value": 14.8,
                "unit": "km/h",
                "unitType": 7
            }
        },
        "relativeHumidity": 71,
        "visibility": {
            "value": 9.7,
            "unit": "km",
            "unitType": 6
        },
        "cloudCover": 100,
        "ceiling": {
            "value": 1128.0,
            "unit": "m",
            "unitType": 5
        },
        "uvIndex": 1,
        "uvIndexPhrase": "Low",
        "precipitationProbability": 51,
        "rainProbability": 51,
        "snowProbability": 0,
        "iceProbability": 0,
        "totalLiquid": {
            "value": 0.3,
            "unit": "mm",
            "unitType": 3
        },
        "rain": {
            "value": 0.3,
            "unit": "mm",
            "unitType": 3
        },
        "snow": {
            "value": 0.0,
            "unit": "cm",
            "unitType": 4
        },
        "ice": {
            "value": 0.0,
            "unit": "mm",
            "unitType": 3
        }
    }
}
]
}

```

Request minute-by-minute weather forecast data

The [Get Minute Forecast API](#) returns minute-by-minute forecasts for a given location for the next 120 minutes. Users can request weather forecasts in intervals of 1, 5 and 15 minutes. The response includes details such as the type of precipitation (including rain, snow, or a mixture of both), start time, and precipitation intensity value (dBZ).

In this example, you'll use the [Get Minute Forecast API](#) to retrieve the minute-by-minute weather forecast at coordinates located in Seattle, WA. The weather forecast is given for the next 120 minutes. Our query requests that the forecast be given at 15-minute intervals, but you can adjust the parameter to be either 1 or 5 minutes.

1. Open the Postman app, click **New**, and select **Request**. Enter a **Request name** for the request. Select the collection you created in the previous section or created a new one, and then select **Save**.
2. Select the **GET** HTTP method in the builder tab and enter the following URL. For this request, and other

requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key.

```
https://atlas.microsoft.com/weather/forecast/minute/json?api-version=1.0&query=47.60357,-122.32945&interval=15&subscription-key={Azure-Maps-Primary-Subscription-key}
```

3. Click the blue **Send** button. The response body contains weather forecast data for the next 120 minutes, in 15-minute intervals.

```
{
  "summary": {
    "briefPhrase60": "No precipitation for at least 60 min",
    "shortPhrase": "No precip for 120 min",
    "briefPhrase": "No precipitation for at least 120 min",
    "longPhrase": "No precipitation for at least 120 min",
    "iconCode": 7
  },
  "intervalSummaries": [
    {
      "startMinute": 0,
      "endMinute": 119,
      "totalMinutes": 120,
      "shortPhrase": "No precip for %MINUTE_VALUE min",
      "briefPhrase": "No precipitation for at least %MINUTE_VALUE min",
      "longPhrase": "No precipitation for at least %MINUTE_VALUE min",
      "iconCode": 7
    }
  ],
  "intervals": [
    {
      "startTime": "2020-10-19T20:51:00+00:00",
      "minute": 0,
      "dbz": 0.0,
      "shortPhrase": "No Precipitation",
      "iconCode": 7,
      "cloudCover": 100
    },
    {
      "startTime": "2020-10-19T21:06:00+00:00",
      "minute": 15,
      "dbz": 0.0,
      "shortPhrase": "No Precipitation",
      "iconCode": 7,
      "cloudCover": 100
    },
    {
      "startTime": "2020-10-19T21:21:00+00:00",
      "minute": 30,
      "dbz": 0.0,
      "shortPhrase": "No Precipitation",
      "iconCode": 7,
      "cloudCover": 100
    },
    {
      "startTime": "2020-10-19T21:36:00+00:00",
      "minute": 45,
      "dbz": 0.0,
      "shortPhrase": "No Precipitation",
      "iconCode": 7,
      "cloudCover": 100
    },
    {
      "startTime": "2020-10-19T21:51:00+00:00",
      "minute": 60,
      "dbz": 0.0,
      "shortPhrase": "No Precipitation",
      "iconCode": 7,
      "cloudCover": 100
    }
  ]
}
```

```
        "shortPhrase": "No Precipitation",
        "iconCode": 7,
        "cloudCover": 100
    },
    {
        "startTime": "2020-10-19T22:06:00+00:00",
        "minute": 75,
        "dbz": 0.0,
        "shortPhrase": "No Precipitation",
        "iconCode": 7,
        "cloudCover": 100
    },
    {
        "startTime": "2020-10-19T22:21:00+00:00",
        "minute": 90,
        "dbz": 0.0,
        "shortPhrase": "No Precipitation",
        "iconCode": 7,
        "cloudCover": 100
    },
    {
        "startTime": "2020-10-19T22:36:00+00:00",
        "minute": 105,
        "dbz": 0.0,
        "shortPhrase": "No Precipitation",
        "iconCode": 7,
        "cloudCover": 100
    }
]
```

Next steps

[Azure Maps Weather services concepts](#)

[Azure Maps Weather services REST API](#)

Request elevation data using the Azure Maps Elevation service

6/1/2021 • 10 minutes to read • [Edit Online](#)

The Azure Maps [Elevation service](#) provides APIs to query elevation data anywhere on the earth's surface. You can request sampled elevation data along paths, within a defined bounding box, or at specific coordinates. Also, you can use the [Render V2 - Get Map Tile API](#) to retrieve elevation data in tile format. The tiles are delivered in GeoTIFF raster format. This article describes how to use Azure Maps Elevation service and the Get Map Tile API to request elevation data. The elevation data can be requested in both GeoJSON and GeoTiff formats.

Prerequisites

1. Make an Azure Maps account in Gen 1 (S1) or Gen 2 pricing tier.
 2. Obtain a primary subscription key, also known as the primary key or the subscription key.

For more information about authentication in Azure Maps, see [Manage Authentication in Azure Maps](#).

This article uses the [Postman](#) application, but you can use a different API development environment.

Request elevation data in raster tile format

To request elevation data in raster tile format, use the [Render V2-Get Map Tile API](#). If the tile can be found, the API returns the tile as a GeoTIFF. Otherwise, the API returns 0. All raster DEM tiles use the geoid (sea level) Earth mode. In this example, we'll request elevation data for Mt. Everest.

TIP

To retrieve a tile at a specific area on the world map, find the correct tile at the appropriate zoom level. Also note that WorldDEM covers the entire global landmass but it doesn't cover oceans. For more information, see [Zoom levels and tile grid](#).

To request elevation data in raster tile format using the Postman app:

1. In the Postman app, select **New**.
 2. In the **Create New** window, select **Collection**.
 3. To rename the collection, right click on your collection, and select **Rename**.
 4. Select **New** again.
 5. In the **Create New** window, select **Request**.
 6. Enter a **Request name** for the request.
 7. Select the collection that you created, and then select **Save**.
 8. On the **Builder** tab, select the **GET** HTTP method and then enter the following URL to request the raster tile.

```
https://atlas.microsoft.com/map/tile?subscription-key={Azure-Maps-Primary-Subscription-key}&api-version=2.0&tilesetId=microsoft.dem&zoom=13&x=6074&y=3432
```

IMPORTANT

For this request, and other requests mentioned in this article, replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key.

9. Select the **Send** button.

You should receive the raster tile that contains the elevation data in GeoTIFF format. Each pixel within the raster tile raw data is of type `float`. The value of each pixel represents the elevation height in meters.

Request elevation data in GeoJSON format

To request elevation data in GeoJSON format, use the Elevation service APIs. This section describes each of these APIs:

- [Get Data for Points](#)
- [Post Data for Points](#)
- [Get Data for Polyline](#)
- [Post Data for Polyline](#)
- [Get Data for Bounding Box](#)

IMPORTANT

When no data can be returned, all APIs return **0**.

Request elevation data for points

In this example, we'll use the [Get Data for Points API](#) to request elevation data at Mt. Everest and Chamlang mountains. Then, we'll use the [Post Data for Points API](#) to request elevation data using the same two points. Latitudes and longitudes in the URL are expected to be in WGS84 (World Geodetic System) decimal degree.

IMPORTANT

The URL character length limit is 2048, so it's not possible to pass more than 100 coordinates as a pipeline-delimited string in a URL GET request. If you intend to pass more than 100 coordinates as a pipeline delimited string, use the Post Data for Points API.

To create the request:

1. In the Postman app, select **New** again.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name** for the request.
4. Select the collection that you previously created, and then select **Save**.
5. On the **Builder** tab, select the **GET** HTTP method, and then enter the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key):

```
https://atlas.microsoft.com/elevation/point/json?subscription-key={Azure-Maps-Primary-Subscription-key}&api-version=1.0&points=-73.998672,40.714728|150.644,-34.397
```

6. Select the **Send** button. You'll receive the following JSON response:

```
{  
  "data": [  
    {  
      "coordinate": {  
        "latitude": 40.714728,  
        "longitude": -73.998672  
      },  
      "elevationInMeter": 12.142355447638208  
    },  
    {  
      "coordinate": {  
        "latitude": -34.397,  
        "longitude": 150.644  
      },  
      "elevationInMeter": 384.47041445517846  
    }  
  ]  
}
```

7. Now, we'll call the [Post Data for Points API](#) to get elevation data for the same two points. On the **Builder** tab, select the **POST** HTTP method and then enter the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key):

```
https://atlas.microsoft.com/elevation/point/json?subscription-key={Azure-Maps-Primary-Subscription-key}&api-version=1.0
```

8. In the **Headers** field of the **POST** request, set `Content-Type` to `application/json`.

9. In the **Body** field, provide the following coordinate point information:

```
[  
  {  
    "lon": -73.998672,  
    "lat": 40.714728  
  },  
  {  
    "lon": 150.644,  
    "lat": -34.397  
  }  
]
```

10. Select **Send**.

Request elevation data samples along a Polyline

In this example, we'll use the [Get Data for Polyline API](#) to request five equally spaced samples of elevation data along a straight line between coordinates at Mt. Everest and Chamlang mountains. Both coordinates must be defined in longitude/latitude format. If you don't specify a value for the `samples` parameter, the number of samples defaults to 10. The maximum number of samples is 2,000.

Then, we'll use the Get Data for Polyline API to request three equally spaced samples of elevation data along a path. We'll define the precise location for the samples by passing in three longitude/latitude coordinate pairs.

Finally, we'll use the [Post Data For Polyline API](#) to request elevation data at the same three equally spaced

samples.

Latitudes and longitudes in the URL are expected to be in WGS84 (World Geodetic System) decimal degree.

IMPORTANT

The URL character length limit is 2048, so it's not possible to pass more than 100 coordinates as a pipeline-delimited string in a URL GET request. If you intend to pass more than 100 coordinates as a pipeline delimited string, use the Post Data For Points API.

To create the request:

1. In the Postman app, select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name**, and then select a collection.
4. Select **Save**.
5. On the **Builder** tab, select the **GET** HTTP method, and then enter the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key):

```
https://atlas.microsoft.com/elevation/line/json?api-version=1.0&subscription-key={Azure-Maps-Primary-Subscription-key}&lines=-73.998672,40.714728|150.644,-34.397&samples=5
```

6. Select the **Send** button. You'll receive the following JSON response:

```
{
  "data": [
    {
      "coordinate": {
        "latitude": 40.714728,
        "longitude": -73.998672
      },
      "elevationInMeter": 12.14236
    },
    {
      "coordinate": {
        "latitude": 21.936796000000001,
        "longitude": -17.838003999999998
      },
      "elevationInMeter": 0.0
    },
    {
      "coordinate": {
        "latitude": 3.1588640000000012,
        "longitude": 38.322664000000003
      },
      "elevationInMeter": 598.66943
    },
    {
      "coordinate": {
        "latitude": -15.619067999999999,
        "longitude": 94.483332000000019
      },
      "elevationInMeter": 0.0
    },
    {
      "coordinate": {
        "latitude": -34.397,
        "longitude": 150.644
      },
      "elevationInMeter": 384.47041
    }
  ]
}
```

7. Now, we'll request three samples of elevation data along a path between coordinates at Mount Everest, Chamlang, and Jannu mountains. In the **Params** field, enter the following coordinate array for the value of the `lines` query key.

```
86.9797222, 27.775 | 86.9252778, 27.9880556 | 88.0444444, 27.6822222
```

8. Change the `samples` query key value to `3`. The image below shows the new values.

GET https://atlas.microsoft.com/elevation/line?api-version=1.0&subscription-key=XXXXXXXXXXXXXXXXXXXXXX

Params (6) Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	KEY	VALUE
<input checked="" type="checkbox"/>	api-version	1.0
<input checked="" type="checkbox"/>	subscription-key	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
<input checked="" type="checkbox"/>	lines	86.9797222, 27.775 86.9252778, 27.9880556 88.0444444, 27.6822222
<input checked="" type="checkbox"/>	samples	3
	Key	Value

9. Select **Send**. You'll receive the following JSON response:

```
{
  "data": [
    {
      "coordinate": {
        "latitude": 27.775,
        "longitude": 86.9797222
      },
      "elevationInMeter": 7116.0348851572589
    },
    {
      "coordinate": {
        "latitude": 27.737403546316028,
        "longitude": 87.411180791156454
      },
      "elevationInMeter": 1798.6945512521534
    },
    {
      "coordinate": {
        "latitude": 27.682222199999998,
        "longitude": 88.0444444
      },
      "elevationInMeter": 7016.9372013588072
    }
  ]
}
```

10. Now, we'll call the [Post Data For Polyline API](#) to get elevation data for the same three points. On the **Builder** tab, select the POST HTTP method, and then enter the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key):

```
https://atlas.microsoft.com/elevation/line/json?api-version=1.0&subscription-key={Azure-Maps-Primary-Subscription-key}&samples=5
```

11. In the **Headers** field of the POST request, set `Content-Type` to `application/json`.

12. In the **Body** field, provide the following coordinate point information.

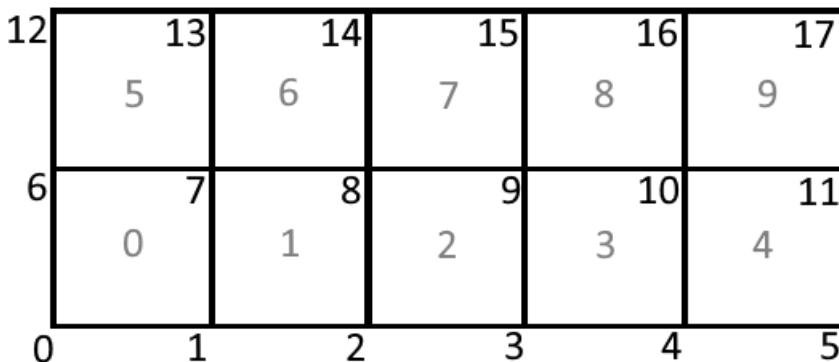
```
[
  {
    "lon": 86.9797222,
    "lat": 27.775
  },
  {
    "lon": 86.9252778,
    "lat": 27.9880556
  },
  {
    "lon": 88.0444444,
    "lat": 27.6822222
  }
]
```

13. Select **Send**.

Request elevation data by Bounding Box

Now we'll use the [Get Data for Bounding Box](#) to request elevation data near Mt. Rainier in Washington state. The elevation data will be returned at equally spaced locations within a bounding box. The bounding area is defined by two sets of latitude/longitude coordinates (south latitude, west longitude | north latitude, east longitude) and is divided into rows and columns. The edges of the bounding box account for two of the rows and two of the columns. Elevations are returned for the grid vertices created at row and column intersections. Up to 2000 elevations can be returned in a single request.

In this example, we'll specify `rows=3` and `columns=6`. The response returns 18 elevation values. In the following diagram, the elevation values are ordered starting with the southwest corner, and then continue west to east and south to north. The elevation points are numbered in the order that they're returned.



To create the request:

1. In the Postman app, select **New**.
2. In the **Create New** window, select **Request**.
3. Enter a **Request name**, and then select a collection.
4. Select **Save**.
5. On the **Builder** tab, select the **GET** HTTP method, and then enter the following URL (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key):

```
https://atlas.microsoft.com/elevation/lattice/json?subscription-key={Azure-Maps-Primary-Subscription-key}&api-version=1.0&bounds=-121.66853362143818, 46.84646479863713,-121.65853362143818, 46.85646479863713&rows=2&columns=3
```

6. Select **Send**. The response returns 18 elevation data samples, one for each vertex of the grid.

```
{
  "data": [
    {
      "coordinate": {
        "latitude": 46.846464798637129,
        "longitude": -121.66853362143819
      },
      "elevationInMeter": 2298.6581875651746
    },
    {
      "coordinate": {
        "latitude": 46.846464798637129,
        "longitude": -121.66653362143819
      },
      "elevationInMeter": 2306.3980756609963
    },
    {
      "coordinate": {
        "latitude": 46.846464798637129,
        "longitude": -121.66453362143818
      },
      "elevationInMeter": 2279.3385479564113
    },
    {
      "coordinate": {
        "latitude": 46.846464798637129,
        "longitude": -121.66253362143819
      },
      "elevationInMeter": 2233.1549264690366
    },
    {
      "coordinate": {
        "latitude": 46.846464798637129,
        "longitude": -121.66053362143818
      },
      "elevationInMeter": 2196.4485923541492
    },
    {
      "coordinate": {
        "latitude": 46.846464798637129,
        "longitude": -121.65853362143818
      },
      "elevationInMeter": 2133.1756767157253
    },
    {
      "coordinate": {
        "latitude": 46.849798131970459,
        "longitude": -121.66853362143819
      },
      "elevationInMeter": 2345.3227848228803
    },
    {
      "coordinate": {
        "latitude": 46.849798131970459,
        "longitude": -121.66653362143819
      },
      "elevationInMeter": 2292.2449195443587
    },
    {
      "coordinate": {
        "latitude": 46.849798131970459,
        "longitude": -121.66453362143818
      },
      "elevationInMeter": 2270.5867788258074
    },
    {
      "coordinate": {
        "latitude": 46.849798131970459,
        "longitude": -121.66253362143819
      },
      "elevationInMeter": 2233.1549264690366
    },
    {
      "coordinate": {
        "latitude": 46.849798131970459,
        "longitude": -121.66053362143818
      },
      "elevationInMeter": 2196.4485923541492
    },
    {
      "coordinate": {
        "latitude": 46.849798131970459,
        "longitude": -121.65853362143818
      },
      "elevationInMeter": 2133.1756767157253
    },
    {
      "coordinate": {
        "latitude": 46.849798131970459,
        "longitude": -121.65653362143819
      },
      "elevationInMeter": 2096.3385479564113
    },
    {
      "coordinate": {
        "latitude": 46.849798131970459,
        "longitude": -121.65453362143818
      },
      "elevationInMeter": 2059.2227848228803
    },
    {
      "coordinate": {
        "latitude": 46.849798131970459,
        "longitude": -121.65253362143819
      },
      "elevationInMeter": 2022.1069195443587
    },
    {
      "coordinate": {
        "latitude": 46.849798131970459,
        "longitude": -121.65053362143818
      },
      "elevationInMeter": 1985.0
    }
  ]
}
```

```
        "coordinate": {
            "latitude": 46.849798131970459,
            "longitude": -121.66253362143819
        },
        "elevationInMeter": 2296.8311427390604
    },
    {
        "coordinate": {
            "latitude": 46.849798131970459,
            "longitude": -121.66053362143818
        },
        "elevationInMeter": 2266.0729430891065
    },
    {
        "coordinate": {
            "latitude": 46.849798131970459,
            "longitude": -121.65853362143818
        },
        "elevationInMeter": 2242.216346631234
    },
    {
        "coordinate": {
            "latitude": 46.8531314653038,
            "longitude": -121.66853362143819
        },
        "elevationInMeter": 2378.460838833359
    },
    {
        "coordinate": {
            "latitude": 46.8531314653038,
            "longitude": -121.66653362143819
        },
        "elevationInMeter": 2327.6761137260387
    },
    {
        "coordinate": {
            "latitude": 46.8531314653038,
            "longitude": -121.66453362143818
        },
        "elevationInMeter": 2208.3782743402949
    },
    {
        "coordinate": {
            "latitude": 46.8531314653038,
            "longitude": -121.66253362143819
        },
        "elevationInMeter": 2106.9526472760981
    },
    {
        "coordinate": {
            "latitude": 46.8531314653038,
            "longitude": -121.66053362143818
        },
        "elevationInMeter": 2054.3270174034078
    },
    {
        "coordinate": {
            "latitude": 46.8531314653038,
            "longitude": -121.65853362143818
        },
        "elevationInMeter": 2030.6438331110671
    },
    {
        "coordinate": {
            "latitude": 46.856464798637127,
            "longitude": -121.66853362143819
        },
        "elevationInMeter": 2318.753153399402
    }
}
```

```
        },
        {
            "coordinate": {
                "latitude": 46.856464798637127,
                "longitude": -121.66653362143819
            },
            "elevationInMeter": 2253.88875188271
        },
        {
            "coordinate": {
                "latitude": 46.856464798637127,
                "longitude": -121.66453362143818
            },
            "elevationInMeter": 2136.6145845357587
        },
        {
            "coordinate": {
                "latitude": 46.856464798637127,
                "longitude": -121.66253362143819
            },
            "elevationInMeter": 2073.6734467948486
        },
        {
            "coordinate": {
                "latitude": 46.856464798637127,
                "longitude": -121.66053362143818
            },
            "elevationInMeter": 2042.994055784251
        },
        {
            "coordinate": {
                "latitude": 46.856464798637127,
                "longitude": -121.65853362143818
            },
            "elevationInMeter": 1988.3631481900356
        }
    ]
}
```

Samples: Use Elevation service APIs in Azure Maps Control

Get elevation data by coordinate position

The following sample webpage describes how to use the map control to display elevation data at a coordinate point. When the user drags the marker, the map displays the elevation data in a pop-up window.

<https://codepen.io/azuremaps/embed/c840b510e113ba7cb32809591d5f96a2?height=500&theme-id=default&default-tab=js,result&editable=true&rerun-position=hidden&>

Get elevation data by bounding box

The following sample webpage describes how to use the map control to display elevation data contained within a bounding box. The user defines the bounding box by selecting the **square** icon in the upper-left corner, and then drawing the square anywhere on the map. The map control then renders the elevation data in accordance with the colors that are specified in the key that's located in the upper-right corner.

<https://codepen.io/azuremaps/embed/619c888c70089c3350a3e95d499f3e48?height=500&theme-id=default&default-tab=js,result&rerun-position=hidden&>

Get elevation data by Polyline path

The following sample webpage describes how to use the map control to display elevation data along a path. The user defines the path by selecting the **Polyline** icon in the upper-left corner, and then drawing the Polyline on

the map. The map control then renders the elevation data in colors that are specified in the key located in the upper-right corner.

<https://codepen.io/azuremaps/embed/7bee08e5cb13d05cb0a11636b60f14ca?height=500&theme-id=default&default-tab=js,result&editable=true&rerun-position=hidden&>

Next steps

To further explore the Azure Maps Elevation APIs, see:

[Elevation - Get Data for Lat Long Coordinates](#)

[Elevation - Get Data for Bounding Box](#)

[Elevation - Get Data for Polyline](#)

[Render V2 – Get Map Tile](#)

For a complete list of Azure Maps REST APIs, see:

[Azure Maps REST APIs](#)

Use the Azure Maps map control

3/23/2021 • 6 minutes to read • [Edit Online](#)

The Map Control client-side JavaScript library allows you to render maps and embedded Azure Maps functionality into your web or mobile application.

This documentation uses the Azure Maps Web SDK, however the Azure Maps services can be used with any map control. [Here](#) are some popular open-source map controls that the Azure Maps team has created plugin's for.

Prerequisites

To use the Map Control in a web page, you must have one of the following prerequisites:

- [Make an Azure Maps account](#) and [obtain a primary subscription key](#), also known as the primary key or the subscription key.
- Obtain your Azure Active Directory (AAD) credentials with [authentication options](#).

Create a new map in a web page

You can embed a map in a web page by using the Map Control client-side JavaScript library.

1. Create a new HTML file.
2. Load in the Azure Maps Web SDK. You can choose one of two options:

- Use the globally hosted CDN version of the Azure Maps Web SDK by adding references to the JavaScript and stylesheet in the `<head>` element of the HTML file:

```
<link rel="stylesheet"
      href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css">
<script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>
```

- Load the Azure Maps Web SDK source code locally using the [azure-maps-control](#) NPM package and host it with your app. This package also includes TypeScript definitions.

```
npm install azure-maps-control
```

Then add references to the Azure Maps stylesheet to the `<head>` element of the file:

```
<link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css"
      type="text/css" />
```

NOTE

TypeScript definitions can be imported into your application by adding the following code:

```
import * as atlas from 'azure-maps-control';
```

3. To render the map so that it fills the full body of the page, add the following `<style>` element to the `<head>` element.

```
<style>
    html, body {
        margin: 0;
    }

    #myMap {
        height: 100vh;
        width: 100vw;
    }
</style>
```

4. In the body of the page, add a `<div>` element and give it an `id` of **myMap**.

```
<body onload="InitMap()">
    <div id="myMap"></div>
</body>
```

5. Now, we'll initialize the map control. In order to authenticate the control, you'll either need to own an Azure Maps subscription key or use Azure Active Directory (AAD) credentials with [authentication options](#).

If you're using a subscription key for authentication, copy and paste the following script element inside the `<head>` element, and below the first `<script>` element. Replace `<Your Azure Maps Key>` with your Azure Maps primary subscription key.

```
<script type="text/javascript">
    function InitMap()
    {
        var map = new atlas.Map('myMap', {
            center: [-122.33, 47.6],
            zoom: 12,
            language: 'en-US',
            authOptions: {
                authType: 'subscriptionKey',
                subscriptionKey: '<Your Azure Maps Key>'
            }
        });
    }
</script>
```

If you're using Azure Active Directory (AAD) for authentication, copy and paste the following script element inside the `<head>` element, and below the first `<script>` element.

```

<script type="text/javascript">
    function InitMap()
    {
        var map = new atlas.Map('myMap', {
            center: [-122.33, 47.6],
            zoom: 12,
            language: 'en-US',
            authOptions: {
                authType: 'aad',
                clientId: '<Your AAD Client Id>',
                aadAppId: '<Your AAD App Id>',
                aadTenant: '<Your AAD Tenant Id>'
            }
        });
    }
</script>

```

For more information about authentication with Azure Maps, see the [Authentication with Azure Maps](#) document. Also, a list of samples showing how to integrate Azure Active Directory (AAD) with Azure Maps can be found [here](#).

TIP

In this example, we've passed in the `id` of the map `<div>`. Another way to do this is to pass in the `HTMLElement` object by passing `document.getElementById('myMap')` as the first parameter.

6. Optionally, you may find it helpful to add the following `meta` elements to the `head` element of the page:

```

<!-- Ensures that IE and Edge uses the latest version and doesn't emulate an older version -->
<meta http-equiv="x-ua-compatible" content="IE=Edge">

<!-- Ensures the web page looks good on all screen sizes. -->
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

```

7. Putting it all together, your HTML file should look something like the following markup:

```

<!DOCTYPE html>
<html>
<head>
    <title></title>

    <meta charset="utf-8">

    <!-- Ensures that IE and Edge uses the latest version and doesn't emulate an older version -->
    <meta http-equiv="x-ua-compatible" content="IE=Edge">

    <!-- Ensures the web page looks good on all screen sizes. -->
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet"
        href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css">
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>

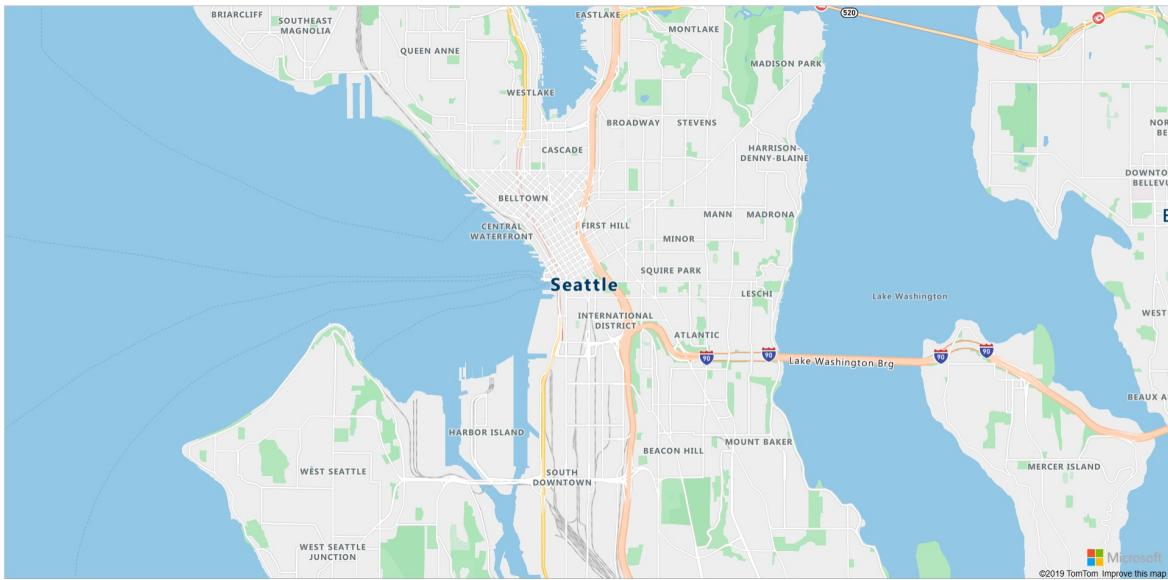
    <script type="text/javascript">
        //Create an instance of the map control and set some options.
        function InitMap()
        {
            var map = new atlas.Map('myMap', {
                center: [-122.33, 47.6],
                zoom: 12,
                language: 'en-US',
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });
        }
    </script>

    <style>
        html, body {
            margin: 0;
        }

        #myMap {
            height: 100vh;
            width: 100vw;
        }
    </style>
</head>
<body onload="InitMap()">
    <div id="myMap"></div>
</body>
</html>

```

8. Open the file in your web browser and view the rendered map. It should look like the image below:



Localizing the map

Azure Maps provides two different ways of setting the language and regional view for the rendered map. The first option is to add this information to the global `atlas` namespace, which will result in all map control instances in your app defaulting to these settings. The following sets the language to French ("fr-FR") and the regional view to "Auto":

```
atlas.setLanguage('fr-FR');
atlas.setView('Auto');
```

The second option is to pass this information into the map options when loading the map like this:

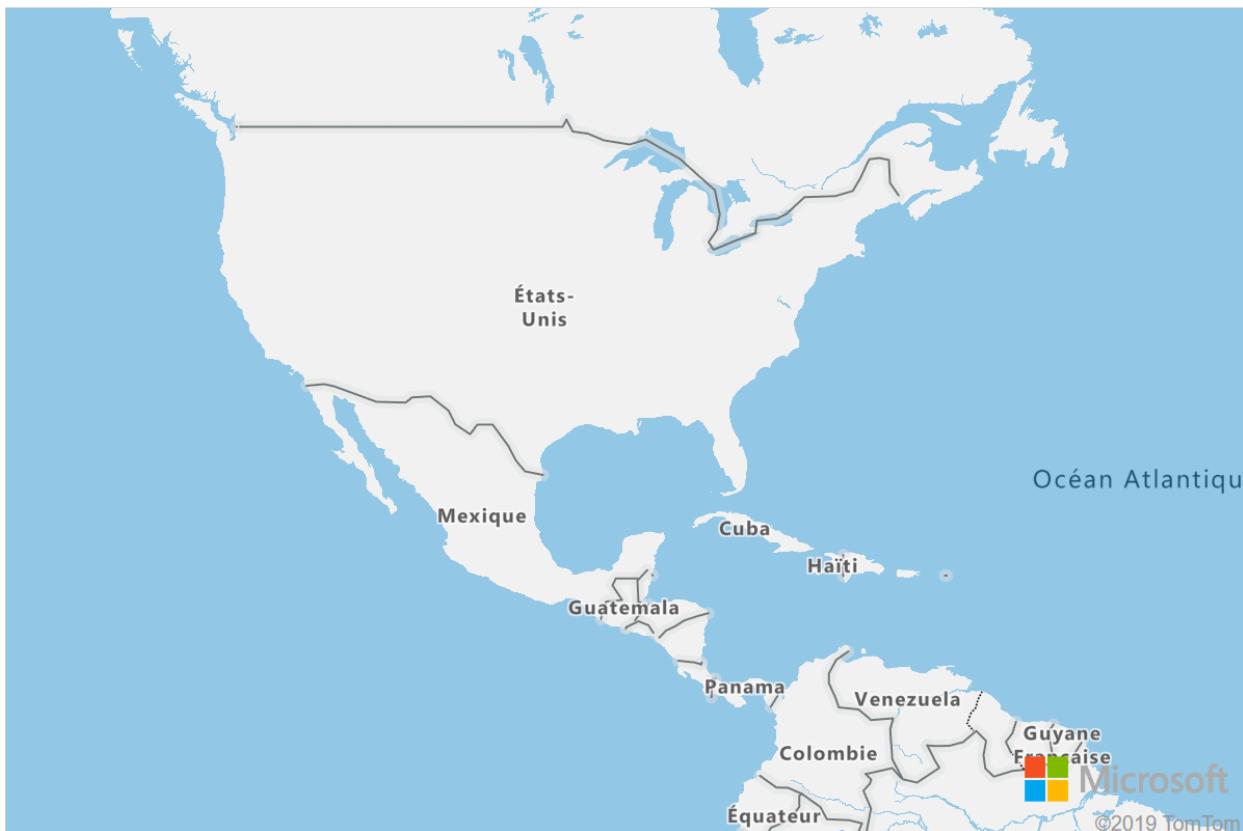
```
map = new atlas.Map('myMap', {
    language: 'fr-FR',
    view: 'Auto',

    authOptions: {
        authType: 'aad',
        clientId: '<Your AAD Client Id>',
        aadAppId: '<Your AAD App Id>',
        aadTenant: '<Your AAD Tenant Id>'
    }
});
```

NOTE

It is possible to load multiple map instances on the same page with different language and region settings. Additionally, these settings can be updated after the map loads using the `setStyle` function of the map.

Here is an example of Azure Maps with the language set to "fr-FR" and the regional view set to "Auto".



A complete list of supported languages and regional views is documented [here](#).

Azure Government cloud support

The Azure Maps Web SDK supports the Azure Government cloud. All JavaScript and CSS URLs used to access the Azure Maps Web SDK remain the same. The following tasks will need to be done to connect to the Azure Government cloud version of the Azure Maps platform.

When using the interactive map control, add the following line of code before creating an instance of the `Map` class.

```
atlas.setDomain('atlas.azure.us');
```

Be sure to use Azure Maps authentication details from the Azure Government cloud platform when authenticating the map and services.

When using the services module, the domain for the services needs to be set when creating an instance of an API URL endpoint. For example, the following code creates an instance of the `SearchURL` class and points the domain to the Azure Government cloud.

```
var searchURL = new atlas.service.SearchURL(pipeline, 'atlas.azure.us');
```

If directly accessing the Azure Maps REST services, change the URL domain to `atlas.azure.us`. For example, if using the search API service, change the URL domain from <https://atlas.microsoft.com/search/> to <https://atlas.azure.us/search/>.

JavaScript frameworks

If developing using a JavaScript framework, one of the following open-source projects may be useful:

- [ng-azure-maps](#) - Angular 10 wrapper around Azure maps.

- [AzureMapsControl.Components](#) - An Azure Maps Blazor component.
- [Azure Maps React Component](#) - A react wrapper for the Azure Maps control.
- [Vue Azure Maps](#) - An Azure Maps component for Vue application.

Next steps

Learn how to create and interact with a map:

[Create a map](#)

Learn how to style a map:

[Choose a map style](#)

Learn best practices and see samples:

[Best practices](#)

[Code samples](#)

For a list of samples showing how to integrate Azure Active Directory (AAD) with Azure Maps, see:

[Azure AD authentication samples](#)

Create a map

11/2/2020 • 5 minutes to read • [Edit Online](#)

This article shows you ways to create a map and animate a map.

Loading a map

To load a map, create a new instance of the [Map class](#). When initializing the map, pass a DIV element ID to render the map and pass a set of options to use when loading the map. If default authentication information isn't specified on the `atlas` namespace, this information will need to be specified in the map options when loading the map. The map loads several resources asynchronously for performance. As such, after creating the map instance, attach a `ready` or `load` event to the map and then add any additional code that interacts with the map to the event handler. The `ready` event fires as soon as the map has enough resources loaded to be interacted with programmatically. The `load` event fires after the initial map view has finished loading completely.

<https://codepen.io/azuremaps/embed/rXdBXx/?height=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

TIP

You can load multiple maps on the same page. Multiple map on the same page may use the same or different authentication and language settings.

Show a single copy of the world

When the map is zoomed out on a wide screen, multiple copies of the world will appear horizontally. This option is great for some scenarios, but for other applications it's desirable to see a single copy of the world. This behavior is implemented by setting the maps `renderWorldCopies` option to `false`.

<https://codepen.io/azuremaps/embed/eqMYpZ/?height=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Map options

When creating a map there, are several different types of options that can be passed in to customize how the map functions as listed below.

- [CameraOptions](#) and [CameraBoundOptions](#) are used to specify the area the map should display.
- [ServiceOptions](#) are used to specify how the map should interact with services that power the map.
- [StyleOptions](#) are used to specify the map should be styled and rendered.
- [UserInteractionOptions](#) are used to specify how the map should react when the user is interacting with the map.

These options can also be updated after the map has been loaded using the `setCamera`, `setServiceOptions`, `setStyle`, and `setUserInteraction` functions.

Controlling the map camera

There are two ways to set the displayed area of the map using the camera of a map. You can set the camera options when loading the map. Or, you can call the `setCamera` option anytime after the map has loaded to programmatically update the map view.

Set the camera

The map camera controls what is displayed in the viewport of the map canvas. Camera options can be passed into the map options when being initialized or passed into the maps `setCamera` function.

```
//Set the camera options when creating the map.  
var map = new atlas.Map('map', {  
    center: [-122.33, 47.6],  
    zoom: 12  
  
    //Additional map options.  
};  
  
//Update the map camera at anytime using setCamera function.  
map.setCamera({  
    center: [-110, 45],  
    zoom: 5  
});
```

In the following code, a [Map object](#) is created and the center and zoom options are set. Map properties, such as center and zoom level, are part of the [CameraOptions](#).

<https://codepen.io/azuremaps/embed/qxKBMN/?height=543&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Set the camera bounds

A bounding box can be used to update the map camera. If the bounding box was calculated from point data, it is often useful to also specify a pixel padding value in the camera options to account for the icon size. This will help ensure that points don't fall off the edge of the map viewport.

```
map.setCamera({  
    bounds: [-122.4, 47.6, -122.3, 47.7],  
    padding: 10  
});
```

In the following code, a [Map object](#) is constructed via `new atlas.Map()`. Map properties such as [CameraBoundsOptions](#) can be defined via [setCamera](#) function of the Map class. Bounds and padding properties are set using `setCamera`.

<https://codepen.io/azuremaps/embed/ZrRbPg/?height=543&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Animate map view

When setting the camera options of the map, [animation options](#) can also be set. These options specify the type of animation and duration it should take to move the camera.

```
map.setCamera({
  center: [-122.33, 47.6],
  zoom: 12,
  duration: 1000,
  type: 'fly'
});
```

In the following code, the first code block creates a map and sets the enter and zoom map styles. In the second code block, a click event handler is created for the animate button. When this button is clicked, the `setCamera` function is called with some random values for the `CameraOptions` and `AnimationOptions`.

<https://codepen.io/azuremaps/embed/WayvbO/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Request transforms

Sometimes it is useful to be able to modify HTTP requests made by the map control. For example:

- Add additional headers to tile requests. This is often done for password protected services.
- Modify URLs to run requests through a proxy service.

The `service options` of the map has a `transformRequest` that can be used to modify all requests made by the map before they are made. The `transformRequest` option is a function that takes in two parameters; a string URL, and a resource type string that indicates what the request is used for. This function must return a `RequestParameters` result.

```
transformRequest: (url: string, resourceType: string) => RequestParameters
```

The following example shows how to use this to modify all requests to the size `https://example.com` by adding a username and password as headers to the request.

```
var map = new atlas.Map('myMap', {
  transformRequest: function (url, resourceType) {
    //Check to see if the request is to the specified endpoint.
    if (url.indexOf('https://examples.com') > -1) {
      //Add custom headers to the request.
      return {
        url: url,
        header: {
          username: 'myUsername',
          password: 'myPassword'
        }
      };
    }

    //Return the URL unchanged by default.
    return { url: url };
  },

  authOptions: {
    authType: 'subscriptionKey',
    subscriptionKey: '<Your Azure Maps Key>'
  }
});
```

Try out the code

Look at the code samples. You can edit the JavaScript code inside the **JS tab** and see the map view changes on the **Result tab**. You can also click **Edit on CodePen**, in the top-right corner, and modify the code in CodePen.

Next steps

Learn more about the classes and methods used in this article:

[Map](#)

[CameraOptions](#)

[AnimationOptions](#)

See code examples to add functionality to your app:

[Change style of the map](#)

[Add controls to the map](#)

[Code samples](#)

Change the style of the map

4/30/2021 • 3 minutes to read • [Edit Online](#)

The map control supports several different map [style options](#) and [base map styles](#). All styles can be set when the map control is being initialized. Or, you can set styles by using the map control's `setStyle` function. This article shows you how to use these style options to customize the map's appearance. Also, you'll learn how to implement the style picker control in your map. The style picker control allows the user to toggle between different base styles.

Set map style options

Style options can be set during web control initialization. Or, you can update style options by calling the map control's `setStyle` function. To see all of the available style options, see [style options](#).

```
//Set the style options when creating the map.  
var map = new atlas.Map('map', {  
    renderWorldCopies: false,  
    showBuildingModels: false,  
    showLogo: true,  
    showFeedbackLink: true,  
    style: 'road'  
  
    //Additional map options.  
});  
  
//Update the style options at anytime using `setStyle` function.  
map.setStyle({  
    renderWorldCopies: true,  
    showBuildingModels: true,  
    showLogo: false,  
    showFeedbackLink: false  
});
```

The following tool shows how the different style options change how the map is rendered. To see the 3D buildings, zoom in close to a major city.

<https://codepen.io/azuremaps/embed/eYNMjPb?height=700&theme-id=0&default-tab=result&rerun-position=hidden&>

Set a base map style

You can also initialize the map control with one of the [base map styles](#) that are available in the Web SDK. You can then use the `setStyle` function to update the base style with a different map style.

Set a base map style on initialization

Base styles of the map control can be set during initialization. In the following code, the `style` option of the map control is set to the [grayscale_dark](#) base map style.

```
var map = new atlas.Map('map', {
    style: 'grayscale_dark',
    //Additional map options
});
```

<https://codepen.io/azuremaps/embed/WKOQRq/%3Fheight=265&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Update the base map style

The base map style can be updated by using the `setStyle` function and setting the `style` option to either change to a different base map style or add additional style options.

```
map.setStyle({ style: 'satellite' });
```

In the following code, after a map instance is loaded, the map style is updated from `grayscale_dark` to `satellite` using the `setStyle` function.

<https://codepen.io/azuremaps/embed/yqXYzY/%3Fheight=265&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Add the style picker control

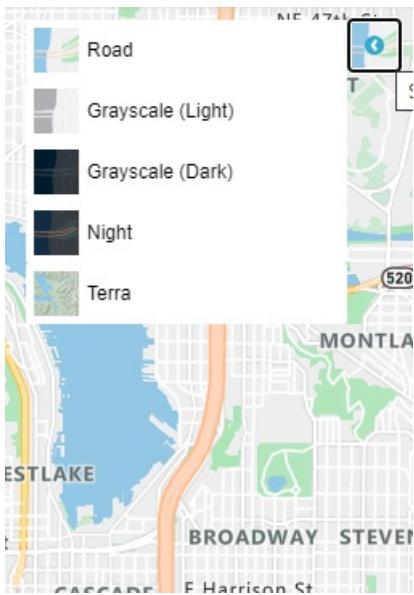
The style picker control provides an easy to use button with flyout panel that can be used by the end user to switch between base styles.

The style picker has two different layout options: `icon` and `list`. Also, the style picker allows you to choose two different style picker control `style` options: `light` and `dark`. In this example, the style picker uses the `icon` layout and displays a select list of base map styles in the form of icons. The style control picker includes the following base set of styles: `["road", "grayscale_light", "grayscale_dark", "night", "road_shaded_relief"]`. For more information on style picker control options, see [Style Control Options](#).

The image below shows the style picker control displayed in `icon` layout.



The image below shows the style picker control displayed in `list` layout.



IMPORTANT

By default the style picker control lists all the styles available under the S0 pricing tier of Azure Maps. If you want to reduce the number of styles in this list, pass an array of the styles you want to appear in the list into the `mapStyle` option of the style picker. If you are using Gen 1 (S1) or Gen 2 pricing tier and want to show all available styles, set the `mapStyles` option of the style picker to `"all"`.

The following code shows you how to override the default `mapStyles` base style list. In this example, we're setting the `mapStyles` option to list which base styles we want to be displayed by the style picker control.

<https://codepen.io/azuremaps/embed/OwggyvG/?height=265&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Next steps

To learn more about the classes and methods used in this article:

[Map](#)

[StyleOptions](#)

[StyleControl](#)

[StyleControlOptions](#)

See the following articles for more code samples to add to your maps:

[Add map controls](#)

[Add a symbol layer](#)

[Add a bubble layer](#)

Add controls to a map

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to add controls to a map. You'll also learn how to create a map with all controls and a [style picker](#).

Add zoom control

A zoom control adds buttons for zooming the map in and out. The following code sample creates an instance of the [ZoomControl](#) class, and adds it the bottom-right corner of the map.

```
//Construct a zoom control and add it to the map.  
map.controls.add(new atlas.control.ZoomControl(), {  
    position: 'bottom-right'  
});
```

Below is the complete running code sample of the above functionality.

<https://codepen.io/azuremaps/embed/WKOQyN/?height=265&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Add pitch control

A pitch control adds buttons for tilting the pitch to map relative to the horizon. The following code sample creates an instance of the [PitchControl](#) class. It adds the PitchControl to top-right corner of the map.

```
//Construct a pitch control and add it to the map.  
map.controls.add(new atlas.control.PitchControl(), {  
    position: 'top-right'  
});
```

Below is the complete running code sample of the above functionality.

<https://codepen.io/azuremaps/embed/xJrwaP/?height=265&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Add compass control

A compass control adds a button for rotating the map. The following code sample creates an instance of the [CompassControl](#) class and adds it the bottom-left corner of the map.

```
//Construct a compass control and add it to the map.  
map.controls.add(new atlas.control.CompassControl(), {  
    position: 'bottom-left'  
});
```

Below is the complete running code sample of the above functionality.

<https://codepen.io/azuremaps/embed/GBEoRb/?height=265&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

A Map with all controls

Multiple controls can be put into an array and added to the map all at once and positioned in the same area of the map to simplify development. The following adds the standard navigation controls to the map using this approach.

```
map.controls.add([
  new atlas.control.ZoomControl(),
  new atlas.control.CompassControl(),
  new atlas.control.PitchControl(),
  new atlas.control.StyleControl()
], {
  position: "top-right"
});
```

The following code sample adds the zoom, compass, pitch, and style picker controls to the top-right corner of the map. Notice how they automatically stack. The order of the control objects in the script dictates the order in which they appear on the map. To change the order of the controls on the map, you can change their order in the array.

<https://codepen.io/azuremaps/embed/qyjbOM/?height=265&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

The style picker control is defined by the [StyleControl](#) class. For more information on using the style picker control, see [choose a map style](#).

Customize controls

Here is a tool to test out the various options for customizing the controls.

<https://codepen.io/azuremaps/embed/LwBZMx/?height=700&theme-id=0&default-tab=result&rerun-position=hidden&>

If you want to create customized navigation controls, create a class that extends from the [atlas.Control](#) class or create an HTML element and position it above the map div. Have this UI control call the maps [setCamera](#) function to move the map.

Next steps

Learn more about the classes and methods used in this article:

[Compass Control](#)

[PitchControl](#)

[StyleControl](#)

[ZoomControl](#)

See the following articles for full code:

[Add a pin](#)

[Add a popup](#)

[Add a line layer](#)

[Add a polygon layer](#)

[Add a bubble layer](#)

Create a data source

6/1/2021 • 8 minutes to read • [Edit Online](#)

The Azure Maps Web SDK stores data in data sources. Using data sources optimizes the data operations for querying and rendering. Currently there are two types of data sources:

- **GeoJSON source:** Manages raw location data in GeoJSON format locally. Good for small to medium data sets (upwards of hundreds of thousands of shapes).
- **Vector tile source:** Loads data formatted as vector tiles for the current map view, based on the maps tiling system. Ideal for large to massive data sets (millions or billions of shapes).

GeoJSON data source

A GeoJSON based data source load and store data locally using the `DataSource` class. GeoJSON data can be manually created or created using the helper classes in the `atlas.data` namespace. The `DataSource` class provides functions to import local or remote GeoJSON files. Remote GeoJSON files must be hosted on a CORs enabled endpoint. The `DataSource` class provides functionality for clustering point data. And, data can easily be added, removed, and updated with the `DataSource` class. The following code shows how GeoJSON data can be created in Azure Maps.

```
//Create raw GeoJSON object.  
var rawGeoJson = {  
    "type": "Feature",  
    "geometry": {  
        "type": "Point",  
        "coordinates": [-100, 45]  
    },  
    "properties": {  
        "custom-property": "value"  
    }  
};  
  
//Create GeoJSON using helper classes (less error prone and less typing).  
var geoJsonClass = new atlas.data.Feature(new atlas.data.Point([-100, 45]), {  
    "custom-property": "value"  
});
```

Once created, data sources can be added to the map through the `map.sources` property, which is a `SourceManager`. The following code shows how to create a `DataSource` and add it to the map.

```
//Create a data source and add it to the map.  
var source = new atlas.source.DataSource();  
map.sources.add(source);
```

The following code shows the different ways GeoJSON data can be added to a `DataSource`.

```
//GeoJsonData in the following code can be a single or array of GeoJSON features or geometries, a GeoJSON
feature collection, or a single or array of atlas.Shape objects.

//Add geoJSON object to data source.
source.add(geoJsonData);

//Load geoJSON data from URL. URL should be on a CORS enabled endpoint.
source.importDataFromUrl(geoJsonUrl);

//Overwrite all data in data source.
source.setShapes(geoJsonData);
```

TIP

Lets say you want to overwrite all data in a `DataSource`. If you make calls to the `clear` then `add` functions, the map might re-render twice, which might cause a bit of a delay. Instead use the `setShapes` function, which will remove and replace all data in the data source and only trigger a single re-render of the map.

Vector tile source

A vector tile source describes how to access a vector tile layer. Use the [VectorTileSource](#) class to instantiate a vector tile source. Vector tile layers are similar to tile layers, but they aren't the same. A tile layer is a raster image. Vector tile layers are a compressed file, in PBF format. This compressed file contains vector map data, and one or more layers. The file can be rendered and styled on the client, based on the style of each layer. The data in a vector tile contain geographic features in the form of points, lines, and polygons. There are several advantages of using vector tile layers instead of raster tile layers:

- A file size of a vector tile is typically much smaller than an equivalent raster tile. As such, less bandwidth is used. It means lower latency, a faster map, and a better user experience.
- Since vector tiles are rendered on the client, they adapt to the resolution of the device they're being displayed on. As a result, the rendered maps appear more well defined, with crystal clear labels.
- Changing the style of the data in the vector maps doesn't require downloading the data again, since the new style can be applied on the client. In contrast, changing the style of a raster tile layer typically requires loading tiles from the server then applying the new style.
- Since the data is delivered in vector form, there's less server-side processing required to prepare the data. As a result, the newer data can be made available faster.

Azure Maps adheres to the [Mapbox Vector Tile Specification](#), an open standard. Azure Maps provides the following vector tiles services as part of the platform:

- Road tiles [documentation](#) | [data format details](#)
- Traffic incidents [documentation](#) | [data format details](#)
- Traffic flow [documentation](#) | [data format details](#)
- Azure Maps Creator also allows custom vector tiles to be created and accessed through the [Render V2-Get Map Tile API](#)

TIP

When using vector or raster image tiles from the Azure Maps render service with the web SDK, you can replace `atlas.microsoft.com` with the placeholder `{azMapsDomain}`. This placeholder will be replaced with the same domain used by the map and will automatically append the same authentication details as well. This greatly simplifies authentication with the render service when using Azure Active Directory authentication.

To display data from a vector tile source on the map, connect the source to one of the data rendering layers. All layers that use a vector source must specify a `sourceLayer` value in the options. The following code loads the Azure Maps traffic flow vector tile service as a vector tile source, then displays it on a map using a line layer. This vector tile source has a single set of data in the source layer called "Traffic flow". The line data in this data set has a property called `traffic_level` that is used in this code to select the color and scale the size of lines.

```
//Create a vector tile source and add it to the map.  
var source = new atlas.source.VectorTileSource(null, {  
    tiles: ['https://azMapsDomain/traffic/flow/tile/pbf?api-version=1.0&style=relative&zoom={z}&x={x}&y={y}'],  
    maxZoom: 22  
});  
map.sources.add(source);  
  
//Create a layer for traffic flow lines.  
var flowLayer = new atlas.layer.LineLayer(source, null, {  
    //The name of the data layer within the data source to pass into this rendering layer.  
    sourceLayer: 'Traffic flow',  
  
    //Color the roads based on the traffic_level property.  
    strokeColor: [  
        'interpolate',  
        ['linear'],  
        ['get', 'traffic_level'],  
        0, 'red',  
        0.33, 'orange',  
        0.66, 'green'  
    ],  
  
    //Scale the width of roads based on the traffic_level property.  
    strokeWidth: [  
        'interpolate',  
        ['linear'],  
        ['get', 'traffic_level'],  
        0, 6,  
        1, 1  
    ]  
});  
  
//Add the traffic flow layer below the labels to make the map clearer.  
map.layers.add(flowLayer, 'labels');
```

<https://codepen.io/azuremaps/embed/wvMXJYJ?height=500&theme-id=default&default-tab=js,result&editable=true&rerun-position=hidden&>

Connecting a data source to a layer

Data is rendered on the map using rendering layers. A single data source can be referenced by one or more rendering layers. The following rendering layers require a data source:

- [Bubble layer](#) - renders point data as scaled circles on the map.
- [Symbol layer](#) - renders point data as icons or text.
- [Heat map layer](#) - renders point data as a density heat map.
- [Line layer](#) - render a line and or render the outline of polygons.
- [Polygon layer](#) - fills the area of a polygon with a solid color or image pattern.

The following code shows how to create a data source, add it to the map, and connect it to a bubble layer. And then, import GeoJSON point data from a remote location into the data source.

```

//Create a data source and add it to the map.
var source = new atlas.source.DataSource();
map.sources.add(source);

//Create a layer that defines how to render points in the data source and add it to the map.
map.layers.add(new atlas.layer.BubbleLayer(source));

//Load the earthquake data.
source.importDataFromUrl('https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/significant_month.geojson');

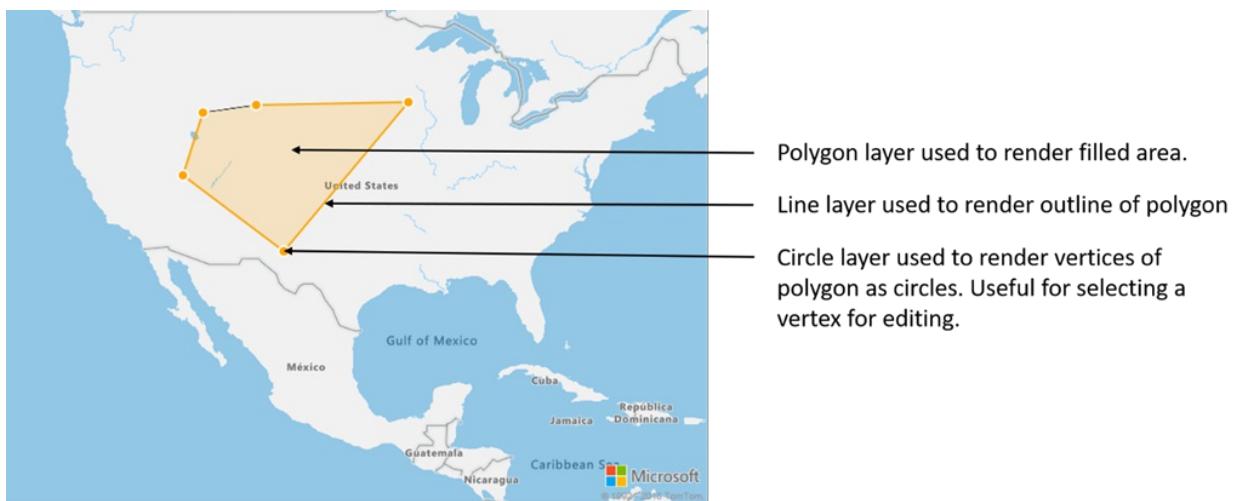
```

There are additional rendering layers that don't connect to these data sources, but they directly load the data for rendering.

- [Image layer](#) - overlays a single image on top of the map and binds its corners to a set of specified coordinates.
- [Tile layer](#) - superimposes a raster tile layer on top of the map.

One data source with multiple layers

Multiple layers can be connected to a single data source. There are many different scenarios in which this option is useful. For example, consider the scenario in which a user draws a polygon. We should render and fill the polygon area as the user adds points to the map. Adding a styled line to outline the polygon makes it easier see the edges of the polygon, as the user draws. To conveniently edit an individual position in the polygon, we may add a handle, like a pin or a marker, above each position.



In most mapping platforms, you would need a polygon object, a line object, and a pin for each position in the polygon. As the polygon is modified, you would need to manually update the line and pins, which can quickly become complex.

With Azure Maps, all you need is a single polygon in a data source as shown in the code below.

```

//Create a data source and add it to the map.
var source = new atlas.source.DataSource();
map.sources.add(source);

//Create a polygon and add it to the data source.
source.add(new atlas.data.Polygon([[[/* Coordinates for polygon */]]]));

//Create a polygon layer to render the filled in area of the polygon.
var polygonLayer = new atlas.layer.PolygonLayer(source, 'myPolygonLayer', {
    fillColor: 'rgba(255,165,0,0.2)'
});

//Create a line layer for greater control of rendering the outline of the polygon.
var lineLayer = new atlas.layer.LineLayer(source, 'myLineLayer', {
    color: 'orange',
    width: 2
});

//Create a bubble layer to render the vertices of the polygon as scaled circles.
var bubbleLayer = new atlas.layer.BubbleLayer(source, 'myBubbleLayer', {
    color: 'orange',
    radius: 5,
    strokeColor: 'white',
    strokeWidth: 2
});

//Add all layers to the map.
map.layers.add([polygonLayer, lineLayer, bubbleLayer]);

```

TIP

When adding layers to the map using the `map.layers.add` function, the ID or instance of an existing layer can be passed in as a second parameter. This would tell map to insert the new layer being added below the existing layer. In addition to passing in a layer ID this method also supports the following values.

- `"labels"` - Inserts the new layer below the map label layers.
- `"transit"` - Inserts the new layer below the map road and transit layers.

Next steps

Learn more about the classes and methods used in this article:

[DataSource](#)

[DataSourceOptions](#)

[VectorTileSource](#)

[VectorTileSourceOptions](#)

See the following articles for more code samples to add to your maps:

[Add a popup](#)

[Use data-driven style expressions](#)

[Add a symbol layer](#)

[Add a bubble layer](#)

[Add a line layer](#)

[Add a polygon layer](#)

[Add a heat map](#)

[Code samples](#)

Add a symbol layer to a map

11/2/2020 • 4 minutes to read • [Edit Online](#)

Connect a symbol to a data source, and use it to render an icon or a text at a given point.

Symbol layers are rendered using WebGL. Use a symbol layer to render large collections of points on the map. Compared to HTML marker, the symbol layer renders a large number of point data on the map, with better performance. However, the symbol layer doesn't support traditional CSS and HTML elements for styling.

TIP

Symbol layers by default will render the coordinates of all geometries in a data source. To limit the layer such that it only renders point geometry features set the `filter` property of the layer to `['==', ['geometry-type'], 'Point']` or `['any', ['==', ['geometry-type'], 'Point'], ['==', ['geometry-type'], 'MultiPoint']]` if you want, you can include MultiPoint features as well.

The maps image sprite manager loads custom images used by the symbol layer. It supports the following image formats:

- JPEG
- PNG
- SVG
- BMP
- GIF (no animations)

Add a symbol layer

Before you can add a symbol layer to the map, you need to take a couple of steps. First, create a data source, and add it to the map. Create a symbol layer. Then, pass in the data source to the symbol layer, to retrieve the data from the data source. Finally, add data into the data source, so that there's something to be rendered.

The code below demonstrates what should be added to the map after it has loaded. This sample renders a single point on the map using a symbol layer.

```
//Create a data source and add it to the map.  
var dataSource = new atlas.source.DataSource();  
map.sources.add(dataSource);  
  
//Create a symbol layer to render icons and/or text at points on the map.  
var layer = new atlas.layer.SymbolLayer(dataSource);  
  
//Add the layer to the map.  
map.layers.add(layer);  
  
//Create a point and add it to the data source.  
dataSource.add(new atlas.data.Point([0, 0]));
```

There are four different types of point data that can be added to the map:

- GeoJSON Point geometry - This object only contains a coordinate of a point and nothing else. The `atlas.data.Point` helper class can be used to easily create these objects.
- GeoJSON MultiPoint geometry - This object contains the coordinates of multiple points and nothing else. The

`atlas.data.MultiPoint` helper class can be used to easily create these objects.

- GeoJSON Feature - This object consists of any GeoJSON geometry and a set of properties that contain metadata associated to the geometry. The `atlas.data.Feature` helper class can be used to easily create these objects.
- `atlas.Shape` class is similar to the GeoJSON feature. Both consist of a GeoJSON geometry and a set of properties that contain metadata associated to the geometry. If a GeoJSON object is added to a data source, it can easily be rendered in a layer. However, if the coordinates property of that GeoJSON object is updated, the data source and map don't change. That's because there's no mechanism in the JSON object to trigger an update. The shape class provides functions for updating the data it contains. When a change is made, the data source and map are automatically notified and updated.

The following code sample creates a GeoJSON Point geometry and passes it into the `atlas.Shape` class to make it easy to update. The center of the map is initially used to render a symbol. A click event is added to the map such that when it fires, the coordinates of the mouse are used with the shapes `setCoordinates` function. The mouse coordinates are recorded at the time of the click event. Then, the `setCoordinates` updates the location of the symbol on the map.

<https://codepen.io/azuremaps/embed/ZqJjRP/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

TIP

By default, symbol layers optimize the rendering of symbols by hiding symbols that overlap. As you zoom in, the hidden symbols become visible. To disable this feature and render all symbols at all times, set the `allowOverlap` property of the `iconOptions` options to `true`.

Add a custom icon to a symbol layer

Symbol layers are rendered using WebGL. As such all resources, such as icon images, must be loaded into the WebGL context. This sample shows how to add a custom icon to the map resources. This icon is then used to render point data with a custom symbol on the map. The `textField` property of the symbol layer requires an expression to be specified. In this case, we want to render the temperature property. Since temperature is a number, it needs to be converted to a string. Additionally we want to append "°F" to it. An expression can be used to do this concatenation: `['concat', ['to-string', ['get', 'temperature']], '°F']`.

<https://codepen.io/azuremaps/embed/WYWRWZ/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

TIP

The Azure Maps web SDK provides several customizable image templates you can use with the symbol layer. For more information, see the [How to use image templates](#) document.

Customize a symbol layer

The symbol layer has many styling options available. Here is a tool to test out these various styling options.

<https://codepen.io/azuremaps/embed/PxVXje/?height=700&theme-id=0&default-tab=result&rerun-position=hidden&>

TIP

When you want to render only text with a symbol layer, you can hide the icon by setting the `image` property of the icon options to `'none'`.

Next steps

Learn more about the classes and methods used in this article:

[SymbolLayer](#)

[SymbolLayerOptions](#)

[IconOptions](#)

[TextOptions](#)

See the following articles for more code samples to add to your maps:

[Create a data source](#)

[Add a popup](#)

[Use data-driven style expressions](#)

[How to use image templates](#)

[Add a line layer](#)

[Add a polygon layer](#)

[Add a bubble layer](#)

[Add HTML Makers](#)

Add a bubble layer to a map

11/2/2020 • 2 minutes to read • [Edit Online](#)

This article shows you how to render point data from a data source as a bubble layer on a map. Bubble layers render points as circles on the map with a fixed pixel radius.

TIP

Bubble layers by default will render the coordinates of all geometries in a data source. To limit the layer such that it only renders point geometry features set the `filter` property of the layer to `['==', ['geometry-type'], 'Point']` or `['any', ['==', ['geometry-type'], 'Point'], ['==', ['geometry-type'], 'MultiPoint']]` if you want to include MultiPoint features as well.

Add a bubble layer

The following code loads an array of points into a data source. Then, it connects the data points are to a [bubble layer](#). The bubble layer renders the radius of each bubble with five pixels and a fill color of white. And, a stroke color of blue, and a stroke width of six pixels.

```
//Add point locations.  
var points = [  
    new atlas.data.Point([-73.985708, 40.75773]),  
    new atlas.data.Point([-73.985600, 40.76542]),  
    new atlas.data.Point([-73.985550, 40.77900]),  
    new atlas.data.Point([-73.975550, 40.74859]),  
    new atlas.data.Point([-73.968900, 40.78859])  
];  
  
//Create a data source and add it to the map.  
var dataSource = new atlas.source.DataSource();  
map.sources.add(dataSource);  
  
//Add multiple points to the data source.  
dataSource.add(points);  
  
//Create a bubble layer to render the filled in area of the circle, and add it to the map.  
map.layers.add(new atlas.layer.BubbleLayer(dataSource, null, {  
    radius: 5,  
    strokeColor: "#4288f7",  
    strokeWidth: 6,  
    color: "white"  
}));
```

Below is the complete running code sample of the above functionality.

<https://codepen.io/azuremaps/embed/mzqaKB/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Show labels with a bubble layer

This code shows you how to use a bubble layer to render a point on the map. And, how to use a symbol layer to render a label. To hide the icon of the symbol layer, set the `image` property of the icon options to `'none'`.

<https://codepen.io/azuremaps/embed/rqbQXy/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Customize a bubble layer

The Bubble layer only has a few styling options. Here is a tool to try them out.

<https://codepen.io/azuremaps/embed/eQxbGm/?height=700&theme-id=0&default-tab=result&rerun-position=hidden&>

Next steps

Learn more about the classes and methods used in this article:

[BubbleLayer](#)

[BubbleLayerOptions](#)

See the following articles for more code samples to add to your maps:

[Create a data source](#)

[Add a symbol layer](#)

[Use data-driven style expressions](#)

[Code samples](#)

Add a popup to the map

11/2/2020 • 8 minutes to read • [Edit Online](#)

This article shows you how to add a popup to a point on a map.

Understand the code

The following code adds a point feature, that has `name` and `description` properties, to the map using a symbol layer. An instance of the [Popup class](#) is created but not displayed. Mouse events are added to the symbol layer to trigger opening and closing the popup. When the marker symbol is hovered, the popup's `position` property is updated with position of the marker, and the `content` option is updated with some HTML that wraps the `name` and `description` properties of the point feature being hovered. The popup is then displayed on the map using its `open` function.

```

//Define an HTML template for a custom popup content layout.
var popupTemplate = '<div class="customInfobox"><div class="name">{name}</div>{description}</div>';

//Create a data source and add it to the map.
var dataSource = new atlas.source.DataSource();
map.sources.add(dataSource);

dataSource.add(new atlas.data.Feature(new atlas.data.Point([-122.1333, 47.63]), {
    name: 'Microsoft Building 41',
    description: '15571 NE 31st St, Redmond, WA 98052'
}));

//Create a layer to render point data.
var symbolLayer = new atlas.layer.SymbolLayer(dataSource);

//Add the polygon and line the symbol layer to the map.
map.layers.add(symbolLayer);

//Create a popup but leave it closed so we can update it and display it later.
popup = new atlas.Popup({
    pixelOffset: [0, -18],
    closeButton: false
});

//Add a hover event to the symbol layer.
map.events.add('mouseover', symbolLayer, function (e) {
    //Make sure that the point exists.
    if (e.shapes && e.shapes.length > 0) {
        var content, coordinate;
        var properties = e.shapes[0].getProperties();
        content = popupTemplate.replace(/{name}/g, properties.name).replace(/{description}/g,
properties.description);
        coordinate = e.shapes[0].getCoordinates();

        popup.setOptions({
            //Update the content of the popup.
            content: content,

            //Update the popup's position with the symbol's coordinate.
            position: coordinate
        });
        //Open the popup.
        popup.open(map);
    }
});

map.events.add('mouseleave', symbolLayer, function () {
    popup.close();
});

```

Below is the complete running code sample of the above functionality.

<https://codepen.io/azuremaps/embed/MPPVz/?height=500&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

Reusing a popup with multiple points

There are cases in which the best approach is to create one popup and reuse it. For example, you may have a large number of points and want to show only one popup at a time. By reusing the popup, the number of DOM elements created by the application is greatly reduced, which can provide better performance. The following sample creates 3-point features. If you click on any of them, a popup will be displayed with the content for that

point feature.

<https://codepen.io/azuremaps/embed/rQbjvK/?height=500&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

Customizing a popup

By default, the popup has a white background, a pointer arrow on the bottom, and a close button in the top-right corner. The following sample changes the background color to black using the `fillColor` option of the popup. The close button is removed by setting the `closeButton` option to false. The HTML content of the popup uses padding of 10 pixels from the edges of the popup. The text is made white, so it shows up nicely on the black background.

<https://codepen.io/azuremaps/embed/ymKgdg/?height=500&theme-id=0&default-tab=result&rerun-position=hidden&>

Add popup templates to the map

Popup templates make it easy to create data driven layouts for popups. The sections below demonstrates the use of various popup templates to generate formatted content using properties of features.

NOTE

By default, all content rendered use the popup template will be sandboxed inside of an iframe as a security feature. However, there are limitations:

- All scripts, forms, pointer lock and top navigation functionality is disabled. Links are allowed to open up in a new tab when clicked.
- Older browsers that don't support the `srcdoc` parameter on iframes will be limited to rendering a small amount of content.

If you trust the data being loaded into the popups and potentially want these scripts loaded into popups be able to access your application, you can disable this by setting the popup templates `sandboxContent` option to false.

String template

The String template replaces placeholders with values of the feature properties. The properties of the feature don't have to be assigned a value of type String. For example, `value1` holds an integer. These values are then passed to the content property of the `popupTemplate`.

The `numberFormat` option specifies the format of the number to display. If the `numberFormat` isn't specified, then the code will use the popup templates date format. The `numberFormat` option formats numbers using the `Number.toLocaleString` function. To format large numbers, consider using the `numberFormat` option with functions from `NumberFormat.format`. For instance, the code snippet below uses `maximumFractionDigits` to limit the number of fraction digits to two.

NOTE

There's only one way in which the String template can render images. First, the String template needs to have an image tag in it. The value being passed to the image tag should be a URL to an image. Then, the String template needs to have `isImage` set to true in the `HyperLinkFormatOptions`. The `isImage` option specifies that the hyperlink is for an image, and the hyperlink will be loaded into an image tag. When the hyperlink is clicked, the image will open.

```

var templateOptions = {
  content: 'This template uses a string template with placeholders.<br/><br/> - Value 1 = {value1}<br/> - Value 2 = {value2/subValue}<br/> - Array value [2] = {arrayValue/2}',
  numberFormat: {
    maximumFractionDigits: 2
  }
};

var feature = new atlas.data.Feature(new atlas.data.Point([0, 0]), {
  title: 'Template 1 - String template',
  value1: 1.2345678,
  value2: {
    subValue: 'Pizza'
  },
  arrayValue: [3, 4, 5, 6]
});

var popup = new atlas.Popup({
  content: atlas.PopupTemplate.applyTemplate(feature.properties, templateOptions),
  position: feature.geometry.coordinates
});

```

PropertyInfo template

The PropertyInfo template displays available properties of the feature. The `label` option specifies the text to display to the user. If `label` isn't specified, then the hyperlink will be displayed. And, if the hyperlink is an image, the value assigned to the "alt" tag will be displayed. The `dateFormat` specifies the format of the date, and if the date format isn't specified, then the date will render as a string. The `hyperlinkFormat` option renders clickable links, similarly, the `email` option can be used to render clickable email addresses.

Before the PropertyInfo template display the properties to the end user, it recursively checks that the properties are indeed defined for that feature. It also ignores displaying style and title properties. For example, it won't display `color`, `size`, `anchor`, `strokeOpacity`, and `visibility`. So, once property path checking is complete in the back-end, the PropertyInfo template shows the content in a table format.

```

var templateOptions = {
  content: [
    {
      propertyPath: 'createDate',
      label: 'Created Date'
    },
    {
      propertyPath: 'dateNumber',
      label: 'Formatted date from number',
      dateFormat: {
        weekday: 'long',
        year: 'numeric',
        month: 'long',
        day: 'numeric',
        timeZone: 'UTC',
        timeZoneName: 'short'
      }
    },
    {
      propertyPath: 'url',
      label: 'Code samples',
      hideLabel: true,
      hyperlinkFormat: {
        label: 'Go to code samples!',
        target: '_blank'
      }
    },
    {
      propertyPath: 'email',
      label: 'Email us',
      hideLabel: true,
      hyperlinkFormat: {
        target: '_blank',
        scheme: 'mailto:'
      }
    }
  ]
};

var feature = new atlas.data.Feature(new atlas.data.Point([0, 0]), {
  title: 'Template 2 - PropertyInfo',
  createDate: new Date(),
  dateNumber: 1569880860542,
  url: 'https://aka.ms/AzureMapsSamples',
  email: 'info@microsoft.com'
}), 

var popup = new atlas.Popup({
  content: atlas.PopupTemplate.applyTemplate(feature.properties, templateOptions),
  position: feature.geometry.coordinates
});

```

Multiple content templates

A feature may also display content using a combination of the String template and the PropertyInfo template. In this case, the String template renders placeholders values on a white background. And, the PropertyInfo template renders a full width image inside a table. The properties in this sample are similar to the properties we explained in the previous samples.

```

var templateOptions = {
    content: [
        'This template has two pieces of content; a string template with placeholders and a array of property info which renders a full width image.<br/><br/> - Value 1 = {value1}<br/> - Value 2 = {value2/subValue}<br/> - Array value [2] = {arrayValue/2}',

        [
            {
                propertyPath: 'imageLink',
                label: 'Image',
                hideImageLabel: true,
                hyperlinkFormat: {
                    isImage: true
                }
            }
        ],
        numberFormat: {
            maximumFractionDigits: 2
        }
    ];
};

var feature = new atlas.data.Feature(new atlas.data.Point([0, 0]), {
    title: 'Template 3 - Multiple content template',
    value1: 1.2345678,
    value2: {
        subValue: 'Pizza'
    },
    arrayValue: [3, 4, 5, 6],
    imageLink: 'https://azurermapscodesamples.azurewebsites.net/common/images/Pike_Market.jpg'
});

var popup = new atlas.Popup({
    content: atlas.PopupTemplate.applyTemplate(feature.properties, templateOptions),
    position: feature.geometry.coordinates
});

```

Points without a defined template

When the Popup template isn't defined to be a String template, a PropertyInfo template, or a combination of both, then it uses the default settings. When the `title` and `description` are the only assigned properties, the popup template shows a white background, a close button in the top-right corner. And, on small and medium screens, it shows an arrow at the bottom. The default settings show inside a table for all properties other than the `title` and the `description`. Even when falling back to the default settings, the popup template can still be manipulated programmatically. For example, users can turn off hyperlink detection and the default settings would still apply to other properties.

Click the points on the map in the CodePen. There is a point on the map for each of the following popup templates: String template, PropertyInfo template, and Multiple content template. There are also three points to show how templates render using the defaulting settings.

<https://codepen.io/azurermaps/embed/dyovrzL/?height=500&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

Reuse popup template

Similar to reusing popup, you can reuse popup templates. This approach is useful when you only want to show one popup template at a time, for multiple points. By reusing the popup template, the number of DOM elements created by the application is reduced, which then improves your application performance. The following sample uses the same popup template for three points. If you click on any of them, a popup will be displayed with the content for that point feature.

<https://codepen.io/azuremaps/embed/WNvzxGw/?height=500&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

Popup events

Popups can be opened, closed, and dragged. The popup class provides events to help developers react to these events. The following sample highlights which events fire when the user opens, closes, or drags the popup.

<https://codepen.io/azuremaps/embed/BXrpvB/?height=500&theme-id=0&default-tab=result&rerun-position=hidden&>

Next steps

Learn more about the classes and methods used in this article:

[Popup](#)

[PopupOptions](#)

[PopupTemplate](#)

See the following great articles for full code samples:

[Add a symbol layer](#)

[Add an HTML marker](#)

[Add a line layer](#)

[Add a polygon layer](#)

Add HTML markers to the map

11/2/2020 • 2 minutes to read • [Edit Online](#)

This article shows you how to add a custom HTML such as an image file to the map as an HTML Marker.

NOTE

HTML Markers do not connect to data sources. Instead position information is added directly to the marker and the marker is added to the maps `markers` property which is a [HtmlMarkerManager](#).

IMPORTANT

Unlike most layers in the Azure Maps Web control which use WebGL for rendering, HTML Markers use traditional DOM elements for rendering. As such, the more HTML markers added to a page, the more DOM elements there are.

Performance can degrade after adding a few hundred HTML markers. For larger data sets consider either clustering your data or using a Symbol or Bubble layer.

Add an HTML marker

The [HtmlMarker](#) class has a default style. You can customize the marker by setting the color and text options of the marker. The default style of the HTML marker class is an SVG template that has a `{color}` and `{text}` placeholder. Set the color and text properties in the HTML marker options for a quick customization.

The following code creates an HTML marker, and sets the color property to "DodgerBlue" and the text property to "10". A popup is attached to the marker and `click` event is used to toggle the visibility of the popup.

```
//Create an HTML marker and add it to the map.
var marker = new atlas.HtmlMarker({
    color: 'DodgerBlue',
    text: '10',
    position: [0, 0],
    popup: new atlas.Popup({
        content: '<div style="padding:10px">Hello World</div>',
        pixelOffset: [0, -30]
    })
});

map.markers.add(marker);

//Add a click event to toggle the popup.
map.events.add('click',marker, () => {
    marker.togglePopup();
});
```

Below is the complete running code sample of the above functionality.

<https://codepen.io/azuremaps/embed/MVoeVw/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Create SVG templated HTML marker

The default `htmlContent` of an Html marker is an SVG template with place folders `{color}` and `{text}` in it. You can create custom SVG strings and add these same placeholders into your SVG such that setting the `color` and `text` options of the marker update these placeholders in your SVG.

<https://codepen.io/azuremaps/embed/LXqMWx/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

TIP

The Azure Maps web SDK provides several SVG image templates that can be used with HTML markers. For more information, see the [How to use image templates](#) document.

Add a CSS styled HTML marker

One of the benefits of HTML markers is that there are many great customizations that can be achieved using CSS. In this sample, the content of the `HtmlMarker` consists of HTML and CSS that create an animated pin that drops into place and pulses.

<https://codepen.io/azuremaps/embed/qJVgMx/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Draggable HTML markers

This sample shows how to make an HTML marker draggable. HTML markers support `drag`, `dragstart`, and `dragend` events.

<https://codepen.io/azuremaps/embed/wQZoEV/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Add mouse events to HTML markers

These samples show how to add mouse and drag events to an HTML marker.

<https://codepen.io/azuremaps/embed/RqOKRz/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Next steps

Learn more about the classes and methods used in this article:

[HtmlMarker](#)

[HtmlMarkerOptions](#)

[HtmlMarkerManager](#)

For more code examples to add to your maps, see the following articles:

[How to use image templates](#)

[Add a symbol layer](#)

[Add a bubble layer](#)

Add a line layer to the map

3/5/2021 • 2 minutes to read • [Edit Online](#)

A line layer can be used to render `LineString` and `MultiLineString` features as paths or routes on the map. A line layer can also be used to render the outline of `Polygon` and `MultiPolygon` features. A data source is connected to a line layer to provide it with data to render.

TIP

Line layers by default will render the coordinates of polygons as well as lines in a data source. To limit the layer such that it only renders `LineString` features set the `filter` property of the layer to `['==', ['geometry-type'], 'LineString']` or `['any', ['==', ['geometry-type'], 'LineString'], ['==', ['geometry-type'], 'MultiLineString']]` if you want to include `MultiLineString` features as well.

The following code shows how to create a line. Add the line to a data source, then render it with a line layer using the `LineLayer` class.

```
//Create a data source and add it to the map.  
var dataSource = new atlas.source.DataSource();  
map.sources.add(dataSource);  
  
//Create a line and add it to the data source.  
dataSource.add(new atlas.data.LineString([[-73.972340, 40.743270], [-74.004420, 40.756800]]));  
  
//Create a line layer to render the line to the map.  
map.layers.add(new atlas.layer.LineLayer(dataSource, null, {  
    strokeColor: 'blue',  
    strokeWidth: 5  
}));
```

Below is the complete running code sample of the above functionality.

<https://codepen.io/azuremaps/embed/qomaKv/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Line layers can be styled using `LineLayerOptions` and `Use data-driven style expressions`.

Add symbols along a line

This sample shows how to add arrow icons along a line on the map. When using a symbol layer, set the "placement" option to "line". This option will render the symbols along the line and rotate the icons (0 degrees = right).

<https://codepen.io/azuremaps/embed/drBJwX/?height=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

TIP

The Azure Maps web SDK provides several customizable image templates you can use with the symbol layer. For more information, see the [How to use image templates](#) document.

Add a stroke gradient to a line

You may apply a single stroke color to a line. You can also fill a line with a gradient of colors to show transition from one line segment to the next line segment. For example, line gradients can be used to represent changes over time and distance, or different temperatures across a connected line of objects. In order to apply this feature to a line, the data source must have the `lineMetrics` option set to `true`, and then a color gradient expression can be passed to the `strokeColor` option of the line. The stroke gradient expression has to reference the `['line-progress']` data expression that exposes the calculated line metrics to the expression.

<https://codepen.io/azuremaps/embed/wZwWJZ/?height=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Customize a line layer

The Line layer has several styling options. Here is a tool to try them out.

<https://codepen.io/azuremaps/embed/GwLrgb/?height=700&theme-id=0&default-tab=result&rerun-position=hidden&>

Next steps

Learn more about the classes and methods used in this article:

[LineLayer](#)

[LineLayerOptions](#)

See the following articles for more code samples to add to your maps:

[Create a data source](#)

[Add a popup](#)

[Use data-driven style expressions](#)

[How to use image templates](#)

[Add a polygon layer](#)

Add a polygon layer to the map

11/2/2020 • 3 minutes to read • [Edit Online](#)

This article shows you how to render the areas of `Polygon` and `MultiPolygon` feature geometries on the map using a polygon layer. The Azure Maps Web SDK also supports the creation of Circle geometries as defined in the [extended GeoJSON schema](#). These circles are transformed into polygons when rendered on the map. All feature geometries can easily be updated when wrapped with the `atlas.Shape` class.

Use a polygon layer

When a polygon layer is connected to a data source and loaded on the map, it renders the area with `Polygon` and `MultiPolygon` features. To create a polygon, add it to a data source, and render it with a polygon layer using the `PolygonLayer` class.

```
//Create a data source and add it to the map.  
var dataSource = new atlas.source.DataSource();  
map.sources.add(dataSource);  
  
//Create a rectangular polygon.  
dataSource.add(new atlas.data.Feature(  
    new atlas.data.Polygon([  
        [-73.98235, 40.76799],  
        [-73.95785, 40.80044],  
        [-73.94928, 40.7968],  
        [-73.97317, 40.76437],  
        [-73.98235, 40.76799]  
    ]])  
);  
  
//Create and add a polygon layer to render the polygon to the map, below the label layer.  
map.layers.add(new atlas.layer.PolygonLayer(dataSource, null,{  
    fillColor: 'red',  
    fillOpacity: 0.7  
}), 'labels');
```

Below is the complete and running sample of the above code.

<https://codepen.io/azuremaps/embed/yKbOvZ/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Use a polygon and line layer together

A line layer is used to render the outline of polygons. The following code sample renders a polygon like the previous example, but now adds a line layer. This line layer is a second layer connected to the data source.

<https://codepen.io/azuremaps/embed/aRyEPy/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Fill a polygon with a pattern

In addition to filling a polygon with a color, you may use an image pattern to fill the polygon. Load an image pattern into the maps image sprite resources and then reference this image with the `fillPattern` property of

the polygon layer.

<https://codepen.io/azuremaps/embed/JzQpYX/%3Fheight=500&theme-id=0&default-tab=js,result&rerun-position=hidden&>

TIP

The Azure Maps web SDK provides several customizable image templates you can use as fill patterns. For more information, see the [How to use image templates](#) document.

Customize a polygon layer

The Polygon layer only has a few styling options. Here is a tool to try them out.

<https://codepen.io/azuremaps/embed/LXvxpG/%3Fheight=700&theme-id=0&default-tab=result&rerun-position=hidden&>

Add a circle to the map

Azure Maps uses an extended version of the GeoJSON schema that provides a definition for circles, as noted [here](#). A circle is rendered on the map by creating a `Point` feature. This `Point` has a `subType` property with a value of `"Circle"` and a `radius` property with a number that represents the radius in meters.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [-122.126986, 47.639754]
  },
  "properties": {
    "subType": "Circle",
    "radius": 100
  }
}
```

The Azure Maps Web SDK converts these `Point` features into `Polygon` features. Then, these features are rendered on the map using polygon and line layers as shown in the following code sample.

<https://codepen.io/azuremaps/embed/PRmzJX/%3Fheight=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Make a geometry easy to update

A `Shape` class wraps a `Geometry` or `Feature` and makes it easy to update and maintain these features. To instantiate a shape variable, pass a geometry or a set of properties to the shape constructor.

```
//Creating a shape by passing in a geometry and a object containing properties.
var shape1 = new atlas.Shape(new atlas.data.Point[0,0], { myProperty: 1 });

//Creating a shape using a feature.
var shape2 = new atlas.Shape(new atlas.data.Feature(new atlas.data.Point[0,0], { myProperty: 1 }));
```

The following code sample shows how to wrap a circle GeoJSON object with a shape class. As the value of the

radius changes in the shape, the circle renders automatically on the map.

<https://codepen.io/azuremaps/embed/ZqMeQY/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Next steps

Learn more about the classes and methods used in this article:

[Polygon](#)

[PolygonLayer](#)

[PolygonLayerOptions](#)

For more code examples to add to your maps, see the following articles:

[Create a data source](#)

[Add a popup](#)

[Use data-driven style expressions](#)

[How to use image templates](#)

[Add a line layer](#)

Additional resources:

[Azure Maps GeoJSON specification extension](#)

Add a polygon extrusion layer to the map

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to use the polygon extrusion layer to render areas of `Polygon` and `MultiPolygon` feature geometries as extruded shapes. The Azure Maps Web SDK supports rendering of Circle geometries as defined in the [extended GeoJSON schema](#). These circles can be transformed into polygons when rendered on the map. All feature geometries may be updated easily when wrapped with the `atlas.Shape` class.

Use a polygon extrusion layer

Connect the [polygon extrusion layer](#) to a data source. Then, loaded it on the map. The polygon extrusion layer renders the areas of a `Polygon` and `MultiPolygon` features as extruded shapes. The `height` and `base` properties of the polygon extrusion layer define the base distance from the ground and height of the extruded shape in **meters**. The following code shows how to create a polygon, add it to a data source, and render it using the Polygon extrusion layer class.

NOTE

The `base` value defined in the polygon extrusion layer should be less than or equal to that of the `height`.

<https://codepen.io/azuremaps/embed/wvvBpvE?height=265&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Add data driven polygons

A choropleth map can be rendered using the polygon extrusion layer. Set the `height` and `fillColor` properties of the extrusion layer to the measurement of the statistical variable in the `Polygon` and `MultiPolygon` feature geometries. The following code sample shows an extruded choropleth map of the United States based on the measurement of the population density by state.

<https://codepen.io/azuremaps/embed/eYYYNox?height=265&theme-id=0&default-tab=result&editable=true&rerun-position=hidden&>

Add a circle to the map

Azure Maps uses an extended version of the GeoJSON schema that provides a definition for circles as noted [here](#). An extruded circle can be rendered on the map by creating a `point` feature with a `subType` property of `Circle` and a numbered `Radius` property representing the radius in **meters**. For example:

```
{  
    "type": "Feature",  
    "geometry": {  
        "type": "Point",  
        "coordinates": [-105.203135, 39.664087]  
    },  
    "properties": {  
        "subType": "Circle",  
        "radius": 1000  
    }  
}
```

The Azure Maps Web SDK converts these `Point` features into `Polygon` features under the hood. These `Point` features can be rendered on the map using polygon extrusion layer as shown in the following code sample.

<https://codepen.io/azuremaps/embed/zYYYrxo?height=265&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Customize a polygon extrusion layer

The Polygon Extrusion layer has several styling options. Here is a tool to try them out.

<https://codepen.io/azuremaps/embed/PoogBRJ/%3Fheight=700&theme-id=0&default-tab=result&rerun-position=hidden&>

Next steps

Learn more about the classes and methods used in this article:

[Polygon](#)

[polygon extrusion layer](#)

Additional resources:

[Azure Maps GeoJSON specification extension](#)

Add a heat map layer

3/5/2021 • 6 minutes to read • [Edit Online](#)

Heat maps, also known as point density maps, are a type of data visualization. They're used to represent the density of data using a range of colors and show the data "hot spots" on a map. Heat maps are a great way to render datasets with large number of points.

Rendering tens of thousands of points as symbols can cover most of the map area. This case likely results in many symbols overlapping. Making it difficult to gain a better understanding of the data. However, visualizing this same dataset as a heat map makes it easy to see the density and the relative density of each data point.

You can use heat maps in many different scenarios, including:

- **Temperature data:** Provides approximations for what the temperature is between two data points.
- **Data for noise sensors:** Shows not only the intensity of the noise where the sensor is, but it can also provide insight into the dissipation over a distance. The noise level at any one site might not be high. If the noise coverage area from multiple sensors overlaps, it's possible that this overlapping area might experience higher noise levels. As such, the overlapped area would be visible in the heat map.
- **GPS trace:** Includes the speed as a weighted height map, where the intensity of each data point is based on the speed. For example, this functionality provides a way to see where a vehicle was speeding.

TIP

Heat map layers by default render the coordinates of all geometries in a data source. To limit the layer so that it only renders point geometry features, set the `filter` property of the layer to `['==', ['geometry-type'], 'Point']`. If you want to include MultiPoint features as well, set the `filter` property of the layer to
`['any', ['==', ['geometry-type'], 'Point'], ['==', ['geometry-type'], 'MultiPoint']]`.

Add a heat map layer

To render a data source of points as a heat map, pass your data source into an instance of the `HeatMapLayer` class, and add it to the map.

In the following code, each heat point has a radius of 10 pixels at all zoom levels. To ensure a better user experience, the heat map is below the label layer. The labels stay clearly visible. The data in this sample is from the [USGS Earthquake Hazards Program](#). It is for significant earthquakes that have occurred in the last 30 days.

```
//Create a data source and add it to the map.
var datasource = new atlas.source.DataSource();
map.sources.add(datasource);

//Load a dataset of points, in this case earthquake data from the USGS.
datasource.importDataFromUrl('https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.geojson');

//Create a heat map and add it to the map.
map.layers.add(new atlas.layer.HeatMapLayer(datasource, null, {
    radius: 10,
    opacity: 0.8
}), 'labels');
```

Here's the complete running code sample of the preceding code.

<https://codepen.io/azuremaps/embed/gQqdQB/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Customize the heat map layer

The previous example customized the heat map by setting the radius and opacity options. The heat map layer provides several options for customization, including:

- `radius` : Defines a pixel radius in which to render each data point. You can set the radius as a fixed number or as an expression. By using an expression, you can scale the radius based on the zoom level, and represent a consistent spatial area on the map (for example, a 5-mile radius).
- `color` : Specifies how the heat map is colorized. A color gradient is a common feature of heat maps. You can achieve the effect with an `interpolate` expression. You can also use a `step` expression for colorizing the heat map, breaking up the density visually into ranges that resemble a contour or radar style map. These color palettes define the colors from the minimum to the maximum density value.

You specify color values for heat maps as an expression on the `heatmap-density` value. The color of area where there's no data is defined at index 0 of the "Interpolation" expression, or the default color of a "Stepped" expression. You can use this value to define a background color. Often, this value is set to transparent, or a semi-transparent black.

Here are examples of color expressions:

INTERPOLATION COLOR EXPRESSION	STEPPED COLOR EXPRESSION
['interpolate', ['linear'], ['heatmap-density'], 0, 'transparent', 0.01, 'purple', 0.5, '#fb00fb', 1, '#00c3ff']	['step', ['heatmap-density'], 'transparent', 0.01, 'navy', 0.25, 'green', 0.50, 'yellow', 0.75, 'red']

- `opacity` : Specifies how opaque or transparent the heat map layer is.
- `intensity` : Applies a multiplier to the weight of each data point to increase the overall intensity of the heatmap. It causes a difference in the weight of data points, making it easier to visualize.
- `weight` : By default, all data points have a weight of 1, and are weighted equally. The weight option acts as

a multiplier, and you can set it as a number or an expression. If a number is set as the weight, it's the equivalence of placing each data point on the map twice. For instance, if the weight is 2, then the density doubles. Setting the weight option to a number renders the heat map in a similar way to using the intensity option.

However, if you use an expression, the weight of each data point can be based on the properties of each data point. For example, suppose each data point represents an earthquake. The magnitude value has been an important metric for each earthquake data point. Earthquakes happen all the time, but most have a low magnitude, and aren't noticed. Use the magnitude value in an expression to assign the weight to each data point. By using the magnitude value to assign the weight, you get a better representation of the significance of earthquakes within the heat map.

- `source` and `source-layer` : Enable you to update the data source.

Here's a tool to test out the different heat map layer options.

<https://codepen.io/azuremaps/embed/WYPaXr/?height=700&theme-id=0&default-tab=result&rerun-position=hidden&>

Consistent zoomable heat map

By default, the radii of data points rendered in the heat map layer have a fixed pixel radius for all zoom levels. As you zoom the map, the data aggregates together and the heat map layer looks different.

Use a `zoom` expression to scale the radius for each zoom level, such that each data point covers the same physical area of the map. This expression makes the heat map layer look more static and consistent. Each zoom level of the map has twice as many pixels vertically and horizontally as the previous zoom level.

Scaling the radius so that it doubles with each zoom level creates a heat map that looks consistent on all zoom levels. To apply this scaling, use `zoom` with a base 2 `exponential interpolation` expression, with the pixel radius set for the minimum zoom level and a scaled radius for the maximum zoom level calculated as

`2 * Math.pow(2, minZoom - maxZoom)` as shown in the following sample. Zoom the map to see how the heat map scales with the zoom level.

<https://codepen.io/azuremaps/embed/OGyMZh/?height=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

TIP

When you enable clustering on the data source, points that are close to one another are grouped together as a clustered point. You can use the point count of each cluster as the weight expression for the heat map. This can significantly reduce the number of points to be rendered. The point count of a cluster is stored in a `point_count` property of the point feature:

```
var layer = new atlas.layer.HeatMapLayer(datasource, null, {
  weight: ['get', 'point_count']
});
```

If the clustering radius is only a few pixels, there would be a small visual difference in the rendering. A larger radius groups more points into each cluster, and improves the performance of the heatmap.

Next steps

Learn more about the classes and methods used in this article:

[HeatMapLayer](#)

[HeatMapLayerOptions](#)

For more code examples to add to your maps, see the following articles:

[Create a data source](#)

[Use data-driven style expressions](#)

Add an image layer to a map

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to overlay an image to a fixed set of coordinates. Here are a few examples of different images types that can be overlaid on maps:

- Images captured from drones
- Building floorplans
- Historical or other specialized map images
- Blueprints of job sites
- Weather radar images

TIP

An [ImageLayer](#) is an easy way to overlay an image on a map. Note that browsers might have difficulty loading a large image. In this case, consider breaking your image up into tiles, and loading them into the map as a [TileLayer](#).

The image layer supports the following image formats:

- JPEG
- PNG
- BMP
- GIF (no animations)

Add an image layer

The following code overlays an image of a [map of Newark, New Jersey, from 1922](#) on the map. An [ImageLayer](#) is created by passing a URL to an image, and coordinates for the four corners in the format

[Top Left Corner, Top Right Corner, Bottom Right Corner, Bottom Left Corner].

```
//Create an image layer and add it to the map.
map.layers.add(new atlas.layer.ImageLayer({
    url: 'newark_nj_1922.jpg',
    coordinates: [
        [-74.22655, 40.773941], //Top Left Corner
        [-74.12544, 40.773941], //Top Right Corner
        [-74.12544, 40.712216], //Bottom Right Corner
        [-74.22655, 40.712216] //Bottom Left Corner
    ]
}));
```

Here's the complete running code sample of the preceding code.

<https://codepen.io/azuremaps/embed/eQodRo/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Import a KML file as ground overlay

This sample demonstrates how to add KML ground overlay information as an image layer on the map. KML ground overlays provide north, south, east, and west coordinates, and a counter-clockwise rotation. But, the

image layer expects coordinates for each corner of the image. The KML ground overlay in this sample is for the Chartres cathedral, and it's sourced from [Wikimedia](#).

The code uses the static `getCoordinatesFromEdges` function from the [ImageLayer](#) class. It calculates the four corners of the image using the north, south, east, west, and rotation information of the KML ground overlay.

<https://codepen.io/azuremaps/embed/EOJgpj/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

TIP

Use the `getPixels` and `getPositions` functions of the image layer class to convert between geographic coordinates of the positioned image layer and the local image pixel coordinates.

Customize an image layer

The image layer has many styling options. Here's a tool to try them out.

<https://codepen.io/azuremaps/embed/RqOGzx/?height=700&theme-id=0&default-tab=result&rerun-position=hidden&>

Next steps

Learn more about the classes and methods used in this article:

[ImageLayer](#)

[ImageLayerOptions](#)

See the following articles for more code samples to add to your maps:

[Add a tile layer](#)

Add a tile layer to a map

3/26/2021 • 4 minutes to read • [Edit Online](#)

This article shows you how to overlay a Tile layer on the map. Tile layers allow you to superimpose images on top of Azure Maps base map tiles. For more information on Azure Maps tiling system, see [Zoom levels and tile grid](#).

A Tile layer loads in tiles from a server. These images can either be pre-rendered or dynamically rendered. Pre-rendered images are stored like any other image on a server using a naming convention that the tile layer understands. Dynamically rendered images use a service to load the images close to real time. There are three different tile service naming conventions supported by Azure Maps [TileLayer](#) class:

- X, Y, Zoom notation - X is the column, Y is the row position of the tile in the tile grid, and the Zoom notation a value based on the zoom level.
- Quadkey notation - Combines x, y, and zoom information into a single string value. This string value becomes a unique identifier for a single tile.
- Bounding Box - Specify an image in the Bounding box coordinates format: `{west},{south},{east},{north}`. This format is commonly used by [web-mapping Services \(WMS\)](#).

TIP

A [TileLayer](#) is a great way to visualize large data sets on the map. Not only can a tile layer be generated from an image, vector data can also be rendered as a tile layer too. By rendering vector data as a tile layer, map control only needs to load the tiles which are smaller in file size than the vector data they represent. This technique is commonly used to render millions of rows of data on the map.

The tile URL passed into a Tile layer must be an http or an https URL to a TileJSON resource or a tile URL template that uses the following parameters:

- `{x}` - X position of the tile. Also needs `{y}` and `{z}`.
- `{y}` - Y position of the tile. Also needs `{x}` and `{z}`.
- `{z}` - Zoom level of the tile. Also needs `{x}` and `{y}`.
- `{quadkey}` - Tile quadkey identifier based on the Bing Maps tile system naming convention.
- `{bbox-epsg-3857}` - A bounding box string with the format `{west},{south},{east},{north}` in the EPSG 3857 Spatial Reference System.
- `{subdomain}` - A placeholder for the subdomain values, if specified the `subdomain` will be added.
- `{azMapsDomain}` - A placeholder to align the domain and authentication of tile requests with the same values used by the map.

Add a tile layer

This sample shows how to create a tile layer that points to a set of tiles. This sample uses the x, y, zoom tiling system. The source of this tile layer is the [OpenSeaMap project](#), which contains crowd sourced nautical charts. When viewing radar data, ideally users would clearly see the labels of cities as they navigate the map. This behavior can be implemented by inserting the tile layer below the `labels` layer.

```
//Create a tile layer and add it to the map below the label layer.
map.layers.add(new atlas.layer.TileLayer({
    tileSize: 256,
    minSourceZoom: 7,
    maxSourceZoom: 17
}), 'labels');
```

Below is the complete running code sample of the above functionality.

<https://codepen.io/azuremaps/embed/BGEQjG?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Add an OGC web-mapping service (WMS)

A web-mapping service (WMS) is an Open Geospatial Consortium (OGC) standard for serving images of map data. There are many open data sets available in this format that you can use with Azure Maps. This type of service can be used with a tile layer if the service supports the `EPSG:3857` coordinate reference system (CRS). When using a WMS service, set the width and height parameters to the same value that is supported by the service, be sure to set this same value in the `tileSize` option. In the formatted URL, set the `BBOX` parameter of the service with the `{bbox-epsg-3857}` placeholder.

The following screenshot shows the above code overlaying a web-mapping service of geological data from the [U.S. Geological Survey \(USGS\)](#) on top of a map, below the labels.

<https://codepen.io/azuremaps/embed/BapjZqr?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Add an OGC web-mapping tile service (WMPS)

A web-mapping tile service (WMPS) is an Open Geospatial Consortium (OGC) standard for serving tiled based overlays for maps. There are many open data sets available in this format that you can use with Azure Maps. This type of service can be used with a tile layer if the service supports the `EPSG:3857` or `GoogleMapsCompatible` coordinate reference system (CRS). When using a WMPS service, set the width and height parameters to the same value that is supported by the service, be sure to set this same value in the `tileSize` option. In the formatted URL, replace the following placeholders accordingly:

- `{TileMatrix}` => `{z}`
- `{RowIndex}` => `{y}`
- `{ColumnIndex}` => `{x}`

The following screenshot shows the above code overlaying a web-mapping tile service of imagery from the [U.S. Geological Survey \(USGS\) National Map](#) on top of a map, below the roads and labels.

<https://codepen.io/azuremaps/embed/BapjZVY?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Customize a tile layer

The tile layer class has many styling options. Here is a tool to try them out.

<https://codepen.io/azuremaps/embed/xQeRWX/?height=700&theme-id=0&default-tab=result&rerun-position=hidden&>

Next steps

Learn more about the classes and methods used in this article:

[TileLayer](#)

[TileLayerOptions](#)

See the following articles for more code samples to add to your maps:

[Add an image layer](#)

Show traffic on the map

3/5/2021 • 2 minutes to read • [Edit Online](#)

There are two types of traffic data available in Azure Maps:

- Incident data - consists of point and line-based data for things such as construction, road closures, and accidents.
- Flow data - provides metrics on the flow of traffic on the roads. Often, traffic flow data is used to color the roads. The colors are based on how much traffic is slowing down the flow, relative to the speed limit, or another metric. There are four values that can be passed into the traffic `flow` option of the map.

FLOW VALUE	DESCRIPTION
<code>none</code>	Doesn't display traffic data on the map
<code>relative</code>	Shows traffic data that's relative to the free-flow speed of the road
<code>relative-delay</code>	Displays areas that are slower than the average expected delay
<code>absolute</code>	Shows the absolute speed of all vehicles on the road

The following code shows how to display traffic data on the map.

```
//Show traffic on the map using the traffic options.  
map.setTraffic({  
    incidents: true,  
    flow: 'relative'  
});
```

Below is the complete running code sample of the above functionality.

<https://codepen.io/azuremaps/embed/WMLRPw/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Traffic overlay options

The following tool lets you switch between the different traffic overlay settings to see how the rendering changes.

<https://codepen.io/azuremaps/embed/RwbPqRY/?height=700&theme-id=0&default-tab=result&rerun-position=hidden&>

Add traffic controls

There are two different traffic controls that can be added to the map. The first control, `TrafficControl`, adds a toggle button that can be used to turn traffic on and off. Options for this control allow you to specify when traffic

settings to use when show traffic. By default this control will display relative traffic flow and incident data, however, you could change this to show absolute traffic flow and no incidents if desired. The second control, `TrafficLegendControl`, adds a traffic flow legend to the map that helps user understand what the color code road highlights mean. This control will only appear on the map when traffic flow data is displayed on the map and will be hidden at all other times.

The following code shows how to add the traffic controls to the map.

```
//Att the traffic control toogle button to the top right corner of the map.  
map.controls.add(new atlas.control.TrafficControl(), { position: 'top-right' });  
  
//Att the traffic legend control to the bottom left corner of the map.  
map.controls.add(new atlas.control.TrafficLegendControl(), { position: 'bottom-left' });
```

<https://codepen.io/azuremaps/embed/ZEWaeLJ?height500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Next steps

Learn more about the classes and methods used in this article:

[Map](#)

[TrafficOptions](#)

Enhance your user experiences:

[Map interaction with mouse events](#)

[Building an accessible map](#)

[Code sample page](#)

Clustering point data

3/26/2021 • 5 minutes to read • [Edit Online](#)

When visualizing many data points on the map, data points may overlap over each other. The overlap may cause the map to become unreadable and difficult to use. Clustering point data is the process of combining point data that are near each other and representing them on the map as a single clustered data point. As the user zooms into the map, the clusters break apart into their individual data points. When you work with large number of data points, use the clustering processes to improve your user experience.

Enabling clustering on a data source

Enable clustering in the `DataSource` class by setting the `cluster` option to `true`. Set `clusterRadius` to select nearby points and combines them into a cluster. The value of `clusterRadius` is in pixels. Use `clusterMaxZoom` to specify a zoom level at which to disable the clustering logic. Here is an example of how to enable clustering in a data source.

```
//Create a data source and enable clustering.  
var datasource = new atlas.source.DataSource(null, {  
    //Tell the data source to cluster point data.  
    cluster: true,  
  
    //The radius in pixels to cluster points together.  
    clusterRadius: 45,  
  
    //The maximum zoom level in which clustering occurs.  
    //If you zoom in more than this, all points are rendered as symbols.  
    clusterMaxZoom: 15  
});
```

TIP

If two data points are close together on the ground, it's possible the cluster will never break apart, no matter how close the user zooms in. To address this, you can set the `clusterMaxZoom` option to disable the clustering logic and simply display everything.

The `DataSource` class provides the following methods related to clustering as well.

METHOD	RETURN TYPE	DESCRIPTION
<code>getClusterChildren(clusterId: number)</code>	<code>Promise<Array<Feature<Geometry, any> Shape>></code>	Retrieves the children of the given cluster on the next zoom level. These children may be a combination of shapes and subclusters. The subclusters will be features with properties matching <code>ClusteredProperties</code> .

METHOD	RETURN TYPE	DESCRIPTION
getClusterExpansionZoom(clusterId: number)	Promise<number>	Calculates a zoom level at which the cluster will start expanding or break apart.
getClusterLeaves(clusterId: number, limit: number, offset: number)	Promise<Array<Feature<Geometry, any> Shape>>	Retrieves all points in a cluster. Set the <code>limit</code> to return a subset of the points, and use the <code>offset</code> to page through the points.

Display clusters using a bubble layer

A bubble layer is a great way to render clustered points. Use expressions to scale the radius and change the color based on the number of points in the cluster. If you display clusters using a bubble layer, then you should use a separate layer to render unclustered data points.

To display the size of the cluster on top of the bubble, use a symbol layer with text, and don't use an icon.

<https://codepen.io/azuremaps/embed/qvzRZY/%3Fheight=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Display clusters using a symbol layer

When visualizing data points, the symbol layer automatically hides symbols that overlap each other to ensure a cleaner user interface. This default behavior might be undesirable if you want to show the data points density on the map. However, these settings can be changed. To display all symbols, set the `allowOverlap` option of the Symbol layers `iconOptions` property to `true`.

Use clustering to show the data points density while keeping a clean user interface. The sample below shows you how to add custom symbols and represent clusters and individual data points using the symbol layer.

<https://codepen.io/azuremaps/embed/Wmqpzz/%3Fheight=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Clustering and the heat maps layer

Heat maps are a great way to display the density of data on the map. This visualization method can handle a large number of data points on its own. If the data points are clustered and the cluster size is used as the weight of the heat map, then the heat map can handle even more data. To achieve this option, set the `weight` option of the heat map layer to `['get', 'point_count']`. When the cluster radius is small, the heat map will look nearly identical to a heat map using the unclustered data points, but it will perform much better. However, the smaller the cluster radius, the more accurate the heat map will be, but with fewer performance benefits.

<https://codepen.io/azuremaps/embed/VRJrgO/%3Fheight=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Mouse events on clustered data points

When mouse events occur on a layer that contains clustered data points, the clustered data point return to the event as a GeoJSON point feature object. This point feature will have the following properties:

PROPERTY NAME	TYPE	DESCRIPTION
<code>cluster</code>	boolean	Indicates if feature represents a cluster.
<code>cluster_id</code>	string	A unique ID for the cluster that can be used with the <code>DataSource</code> <code>getClusterExpansionZoom</code> , <code>getClusterChildren</code> , and <code>getClusterLeaves</code> methods.
<code>point_count</code>	number	The number of points the cluster contains.
<code>point_count_abbreviated</code>	string	A string that abbreviates the <code>point_count</code> value if it's long. (for example, 4,000 becomes 4K)

This example takes a bubble layer that renders cluster points and adds a click event. When the click event triggers, the code calculates and zooms the map to the next zoom level, at which the cluster breaks apart. This functionality is implemented using the `getClusterExpansionZoom` method of the `DataSource` class and the `cluster_id` property of the clicked clustered data point.

<https://codepen.io/azuremaps/embed/moZWeV/?height=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Display cluster area

The point data that a cluster represents is spread over an area. In this sample when the mouse is hovered over a cluster, two main behaviors occur. First, the individual data points contained in the cluster will be used to calculate a convex hull. Then, the convex hull will be displayed on the map to show an area. A convex hull is a polygon that wraps a set of points like an elastic band and can be calculated using the `atlas.math.getConvexHull` method. All points contained in a cluster can be retrieved from the data source using the `getClusterLeaves` method.

<https://codepen.io/azuremaps/embed/QoXqWJ/?height=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Aggregating data in clusters

Often clusters are represented using a symbol with the number of points that are within the cluster. But, sometimes it's desirable to customize the style of clusters with additional metrics. With cluster aggregates, custom properties can be created and populated using an [aggregate expression](#) calculation. Cluster aggregates can be defined in `clusterProperties` option of the `DataSource`.

The following sample uses an aggregate expression. The code calculates a count based on the entity type property of each data point in a cluster. When a user clicks on a cluster, a popup shows with additional information about the cluster.

<https://codepen.io/azuremaps/embed/jgYyRL/?height=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Next steps

Learn more about the classes and methods used in this article:

[DataSource class](#)

[DataSourceOptions object](#)

[atlas.math namespace](#)

See code examples to add functionality to your app:

[Add a bubble layer](#)

[Add a symbol layer](#)

[Add a heat map layer](#)

Data-driven Style Expressions (Web SDK)

3/5/2021 • 31 minutes to read • [Edit Online](#)

Expressions enable you to apply business logic to styling options that observe the properties defined in each shape in a data source. Expressions can filter data in a data source or a layer. Expressions may consist of conditional logic, like if-statements. And, they can be used to manipulate data using: string operators, logical operators, and mathematical operators.

Data-driven styles reduce the amount of code needed to implement business logic around styling. When used with layers, expressions are evaluated at render time on a separate thread. This functionality provides increased performance compared to evaluating business logic on the UI thread.

This video provides an overview of data-driven styling in the Azure Maps Web SDK.

Expressions are represented as JSON arrays. The first element of an expression in the array is a string that specifies the name of the expression operator. For example, "+" or "case". The next elements (if any) are the arguments to the expression. Each argument is either a literal value (a string, number, boolean, or `null`), or another expression array. The following pseudocode defines the basic structure of an expression.

```
[  
    expression_operator,  
    argument0,  
    argument1,  
    ...  
)
```

The Azure Maps Web SDK supports many types of expressions. Expressions can be used on their own or in combination with other expressions.

TYPE OF EXPRESSIONS	DESCRIPTION
Aggregate expression	An expression that defines a calculation that is processed over a set of data and can be used with the <code>clusterProperties</code> option of a DataSource .
Boolean expressions	Boolean expressions provide a set of boolean operators expressions for evaluating boolean comparisons.
Color expressions	Color expressions make it easier to create and manipulate color values.
Conditional expressions	Conditional expressions provide logic operations that are like if-statements.
Data expressions	Provides access to the property data in a feature.
Interpolate and Step expressions	Interpolate and step expressions can be used to calculate values along an interpolated curve or step function.

TYPE OF EXPRESSIONS	DESCRIPTION
Layer specific expressions	Special expressions that are only applicable to a single layer.
Math expressions	Provides mathematical operators to perform data-driven calculations within the expression framework.
String operator expressions	String operator expressions perform conversion operations on strings such as concatenating and converting the case.
Type expressions	Type expressions provide tools for testing and converting different data types like strings, numbers, and boolean values.
Variable binding expressions	Variable binding expressions store the results of a calculation in a variable and referenced elsewhere in an expression multiple times without having to recalculate the stored value.
Zoom expression	Retrieves the current zoom level of the map at render time.

All examples in this document use the following feature to demonstrate different ways in which the different types of expressions can be used.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [-122.13284, 47.63699]
  },
  "properties": {
    "id": 123,
    "entityType": "restaurant",
    "revenue": 12345,
    "subTitle": "Building 40",
    "temperature": 64,
    "title": "Cafeteria",
    "zoneColor": "purple",
    "abcArray": ["a", "b", "c"],
    "array2d": [["a", "b"], ["x", "y"]],
    "_style": {
      "fillColor": "red"
    }
  }
}
```

Data expressions

Data expressions provide access to the property data in a feature.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>['at', number, array]</code>	value	Retrieves an item from an array.
<code>['geometry-type']</code>	string	Gets the feature's geometry type: Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>['get', string]</code>	value	Gets the property value from the current feature's properties. Returns null if the requested property is missing.
<code>['get', string, object]</code>	value	Gets the property value from the properties of the provided object. Returns null if the requested property is missing.
<code>['has', string]</code>	boolean	Determines if the properties of a feature have the specified property.
<code>['has', string, object]</code>	boolean	Determines if the properties of the object have the specified property.
<code>['id']</code>	value	Gets the feature's ID if it has one.
<code>['in', boolean string number, array]</code>	boolean	Determines whether an item exists in an array
<code>['in', substring, string]</code>	boolean	Determines whether a substring exists in a string
<code>['index-of', boolean string number, array string]</code>	number	Returns the first position at which an item can be found in an array or a substring can be found in a string, or -1 if the input cannot be found.
<code>['index-of', boolean string number, array string, number]</code>		Accepts an optional index from where to begin the search.
<code>['length', string array]</code>	number	Gets the length of a string or an array.
<code>['slice', array string, number]</code>	string array	Returns an item from an array or a substring from a string from a specified start index, or between a start index and an end index if set. The return value is inclusive of the start index but not of the end index.
<code>['slice', array string, number, number]</code>		

Examples

Properties of a feature can be accessed directly in an expression by using a `get` expression. This example uses the `zoneColor` value of the feature to specify the color property of a bubble layer.

```
var layer = new atlas.layer.BubbleLayer(datasource, null, {
  color: ['get', 'zoneColor'] //Get the zoneColor value.
});
```

The above example will work fine, if all the point features have the `zoneColor` property. If they don't, the color will likely fall back to "black". To modify the fallback color, use a `case` expression in combination with the `has` expression to check if the property exists. If the property doesn't exist, return a fallback color.

```

var layer = new atlas.layer.BubbleLayer(datasource, null, {
  color: [
    'case', //Use a conditional case expression.

    ['has', 'zoneColor'], //Check to see if feature has a "zoneColor" property
    ['get', 'zoneColor'], //If it does, use it.

    'blue' //If it doesn't, default to blue.
  ]
});
```

Bubble and symbol layers will render the coordinates of all shapes in a data source, by default. This behavior can highlight the vertices of a polygon or a line. The `filter` option of the layer can be used to limit the geometry type of the features it renders, by using a `['geometry-type']` expression within a boolean expression. The following example limits a bubble layer so that only `Point` features are rendered.

```

var layer = new atlas.layer.BubbleLayer(datasource, null, {
  filter: ['==', ['geometry-type'], 'Point']
});
```

The following example allows both `Point` and `MultiPoint` features to be rendered.

```

var layer = new atlas.layer.BubbleLayer(datasource, null, {
  filter: ['any', ['==', ['geometry-type'], 'Point'], ['==', ['geometry-type'], 'MultiPoint']]
});
```

Similarly, the outline of Polygons will render in line layers. To disable this behavior in a line layer, add a filter that only allows `LineString` and `MultiLineString` features.

Here are some additional examples of how to use data expressions:

```

//Get item [2] from an array "properties.abcArray[1]" = "c"
['at', 2, ['get', 'abcArray']]

//Get item [0][1] from a 2D array "properties.array2d[0][1]" = "b"
['at', 1, ['at', 0, ['get', 'array2d']]]

//Check to see if a value is in an array "properties.abcArray.indexOf('a') !== -1" = true
['in', 'a', ['get', 'abcArray']]

//Gets the index of the value 'b' in an array "properties.abcArray.indexOf('b')" = 1
['index-of', 'b', ['get', 'abcArray']]

//Get the length of an array "properties.abcArray.length" = 3
['length', ['get', 'abcArray']]

//Get the value of a subproperty "properties._style.fillColor" = "red"
['get', 'fillColor', ['get', '_style']]

//Check that "fillColor" exists as a subproperty of "_style".
['has', 'fillColor', ['get', '_style']]

//Slice an array starting at index 2 "properties.abcArray.slice(2)" = ['c']
['slice', ['get', 'abcArray'], 2]

//Slice a string from index 0 to index 4 "properties.entityType.slice(0, 4)" = 'rest'
['slice', ['get', 'entityType'], 0, 4]
```

Math expressions

Math expressions provide mathematical operators to perform data-driven calculations within the expression framework.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>['+', number, number, ...]</code>	number	Calculates the sum of the specified numbers.
<code>['-', number]</code>	number	Subtracts 0 by the specified number.
<code>['-', number, number]</code>	number	Subtracts the first numbers by the second number.
<code>['*', number, number, ...]</code>	number	Multiplies the specified numbers together.
<code>['/', number, number]</code>	number	Divides the first number by the second number.
<code>['%', number, number]</code>	number	Calculates the remainder when dividing the first number by the second number.
<code>['^', number, number]</code>	number	Calculates the value of the first value raised to the power of the second number.
<code>['abs', number]</code>	number	Calculates the absolute value of the specified number.
<code>['acos', number]</code>	number	Calculates the arccosine of the specified number.
<code>['asin', number]</code>	number	Calculates the arcsine of the specified number.
<code>['atan', number]</code>	number	Calculates the arctangent of the specified number.
<code>['ceil', number]</code>	number	Rounds the number up to the next whole integer.
<code>['cos', number]</code>	number	Calculates the cos of the specified number.
<code>['e']</code>	number	Returns the mathematical constant <code>e</code> .
<code>['floor', number]</code>	number	Rounds the number down to the previous whole integer.
<code>['ln', number]</code>	number	Calculates the natural logarithm of the specified number.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>['ln2']</code>	number	Returns the mathematical constant <code>ln(2)</code> .
<code>['log10', number]</code>	number	Calculates the base-ten logarithm of the specified number.
<code>['log2', number]</code>	number	Calculates the base-two logarithm of the specified number.
<code>['max', number, number, ...]</code>	number	Calculates the maximum number in the specified set of numbers.
<code>['min', number, number, ...]</code>	number	Calculates the minimum number in the specified set of numbers.
<code>['pi']</code>	number	Returns the mathematical constant <code>PI</code> .
<code>['round', number]</code>	number	Rounds the number to the nearest integer. Halfway values are rounded away from zero. For example, <code>['round', -1.5]</code> evaluates to <code>-2</code> .
<code>['sin', number]</code>	number	Calculates the sine of the specified number.
<code>['sqrt', number]</code>	number	Calculates the square root of the specified number.
<code>['tan', number]</code>	number	Calculates the tangent of the specified number.

Aggregate expression

An aggregate expression defines a calculation that's processed over a set of data and can be used with the `clusterProperties` option of a `DataSource`. The output of these expressions must be a number or a boolean.

An aggregate expression takes in three values: an operator value, and initial value, and an expression to retrieve a property from each feature in a data to apply the aggregate operation on. This expression has the following format:

```
[operator: string, initialValue: boolean | number, mapExpression: Expression]
```

- **operator:** An expression function that's then applied to against all values calculated by the `mapExpression` for each point in the cluster. Supported operators:
 - For numbers: `+`, `*`, `max`, `min`
 - For Booleans: `all`, `any`
- **initialValue:** An initial value in which the first calculated value is aggregated against.
- **mapExpression:** An expression that's applied against each point in the data set.

Examples

If all features in a data set have a `revenue` property, which is a number. Then, the total revenue of all points in a cluster, which are created from the data set, can be calculated. This calculation is done using the following aggregate expression: `['+', 0, ['get', 'revenue']]`

Accumulated expression

The `accumulated` expression gets the value of a cluster property accumulated so far. This can only be used in the `clusterProperties` option of a clustered `DataSource` source.

Usage

```
["accumulated"]
```

Boolean expressions

Boolean expressions provide a set of boolean operators expressions for evaluating boolean comparisons.

When comparing values, the comparison is strictly typed. Values of different types are always considered unequal. Cases where the types are known to be different at parse time are considered invalid and will produce a parse error.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>['!', boolean]</code>	boolean	Logical negation. Returns <code>true</code> if the input is <code>false</code> , and <code>false</code> if the input is <code>true</code> .
<code>['!=', value, value]</code>	boolean	Returns <code>true</code> if the input values are not equal, <code>false</code> otherwise.
<code>['<', value, value]</code>	boolean	Returns <code>true</code> if the first input is strictly less than the second, <code>false</code> otherwise. The arguments are required to be either both strings or both numbers.
<code>['<=', value, value]</code>	boolean	Returns <code>true</code> if the first input is less than or equal to the second, <code>false</code> otherwise. The arguments are required to be either both strings or both numbers.
<code>['==', value, value]</code>	boolean	Returns <code>true</code> if the input values are equal, <code>false</code> otherwise. The arguments are required to be either both strings or both numbers.
<code>['>', value, value]</code>	boolean	Returns <code>true</code> if the first input is strictly greater than the second, <code>false</code> otherwise. The arguments are required to be either both strings or both numbers.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>['>=' value, value]</code>	boolean	Returns <code>true</code> if the first input is greater than or equal to the second, <code>false</code> otherwise. The arguments are required to be either both strings or both numbers.
<code>['all', boolean, boolean, ...]</code>	boolean	Returns <code>true</code> if all the inputs are <code>true</code> , <code>false</code> otherwise.
<code>['any', boolean, boolean, ...]</code>	boolean	Returns <code>true</code> if any of the inputs are <code>true</code> , <code>false</code> otherwise.
<code>['within', Polygon MultiPolygon Feature<Polygon MultiPolygon>, ...]</code>	boolean	Returns <code>true</code> if the evaluated feature is fully contained inside a boundary of the input geometry, <code>false</code> otherwise. The input value can be a valid GeoJSON of type <code>Polygon</code> , <code>MultiPolygon</code> , <code>Feature</code> , or <code>FeatureCollection</code> . Supported features for evaluation: - Point: Returns <code>false</code> if a point is on the boundary or falls outside the boundary. - LineString: Returns <code>false</code> if any part of a line falls outside the boundary, the line intersects the boundary, or a line's endpoint is on the boundary.

Conditional expressions

Conditional expressions provide logic operations that are like if-statements.

The following expressions perform conditional logic operations on the input data. For example, the `case` expression provides "if/then/else" logic while the `match` expression is like a "switch-statement".

Case expression

A `case` expression is a type of conditional expression that provides "if/then/else" logic. This type of expression steps through a list of boolean conditions. It returns the output value of the first boolean condition to evaluate to true.

The following pseudocode defines the structure of the `case` expression.

```
[
  'case',
  condition1: boolean,
  output1: value,
  condition2: boolean,
  output2: value,
  ...
  fallback: value
]
```

Example

The following example steps through different boolean conditions until it finds one that evaluates to `true`, and then returns that associated value. If no boolean condition evaluates to `true`, a fallback value will be returned.

```
var layer = new atlas.layer.BubbleLayer(datasource, null, {
  color: [
    'case',
    //Check to see if the first boolean expression is true, and if it is, return its assigned result.
    ['has', 'zoneColor'],
    ['get', 'zoneColor'],
    //Check to see if the second boolean expression is true, and if it is, return its assigned result.
    ['all', ['has', 'temperature'], ['>', ['get', 'temperature'], 100]],
    'red',
    //Specify a default value to return.
    'green'
  ]
});
```

Match expression

A `match` expression is a type of conditional expression that provides switch-statement like logic. The input can be any expression such as `['get', 'entityType']` that returns a string or a number. Each label must be either a single literal value or an array of literal values, whose values must be all strings or all numbers. The input matches if any of the values in the array match. Each label must be unique. If the input type doesn't match the type of the labels, the result will be the fallback value.

The following pseudocode defines the structure of the `match` expression.

```
[
  'match',
  input: number | string,
  label1: number | string | (number | string)[],
  output1: value,
  label2: number | string | (number | string)[],
  output2: value,
  ...,
  fallback: value
]
```

Examples

The following example looks at the `entityType` property of a Point feature in a bubble layer searches for a match. If it finds a match, that specified value is returned or it returns the fallback value.

```
var layer = new atlas.layer.BubbleLayer(datasource, null, {
  color: [
    'match',
    //Get the property to match.
    ['get', 'entityType'],
    //List the values to match and the result to return for each match.
    'restaurant', 'red',
    'park', 'green',
    //Specify a default value to return if no match is found.
    'black'
  ]
});
```

The following example uses an array to list a set of labels that should all return the same value. This approach is much more efficient than listing each label individually. In this case, if the `entityType` property is "restaurant" or "grocery_store", the color "red" will be returned.

```
var layer = new atlas.layer.BubbleLayer(datasource, null, {
  color: [
    'match',
    //Get the property to match.
    ['get', 'entityType'],
    //List the values to match and the result to return for each match.
    ['restaurant', 'grocery_store'], 'red',
    'park', 'green',
    //Specify a default value to return if no match is found.
    'black'
  ]
});
```

Coalesce expression

A `coalesce` expression steps through a set of expressions until the first non-null value is obtained and returns that value.

The following pseudocode defines the structure of the `coalesce` expression.

```
[
  'coalesce',
  value1,
  value2,
  ...
]
```

Example

The following example uses a `coalesce` expression to set the `textField` option of a symbol layer. If the `title` property is missing from the feature or set to `null`, the expression will then try looking for the `subTitle` property, if its missing or `null`, it will then fall back to an empty string.

```
var layer = new atlas.layer.SymbolLayer(datasource, null, {
  textOptions: {
    textField: [
      'coalesce',
      //Try getting the title property.
      ['get', 'title'],
      //If there is no title, try getting the subTitle.
      ['get', 'subTitle'],
      //Default to an empty string.
      ''
    ]
  }
});
```

The following example uses a `coalesce` expression to retrieve the first available image icon available in the map sprite from a list of specified image names.

```

var layer = new atlas.layer.SymbolLayer(datasource, null, {
  iconOptions: {
    image: [
      'coalesce',

      //Try getting the image with id 'missing-image'.
      ['image', 'missing-image'],

      //Specify an image id to fallback to.
      'marker-blue'
    ]
  }
});

```

Type expressions

Type expressions provide tools for testing and converting different data types like strings, numbers, and boolean values.

EXPRESSION	RETURN TYPE	DESCRIPTION
<pre>['array', value] ['array', type: "string" "number" "boolean", value]</pre>	Object[]	Asserts that the input is an array.
<pre>['boolean', value] ["boolean", value, fallback: value, fallback: value, ...]</pre>	boolean	Asserts that the input value is a boolean. If multiple values are provided, each one is evaluated in order until a boolean is obtained. If none of the inputs are booleans, the expression is an error.
<pre>['collator', { 'case-sensitive': boolean, 'diacritic-sensitive': boolean, 'locale': string }]</pre>	collator	Returns a collator for use in locale-dependent comparison operations. The case-sensitive and diacritic-sensitive options default to false. The locale argument specifies the IETF language tag of the locale to use. If none is provided, the default locale is used. If the requested locale is not available, the collator will use a system-defined fallback locale. Use resolved-locale to test the results of locale fallback behavior.
<pre>['literal', array] ['literal', object]</pre>	array object	Returns a literal array or object value. Use this expression to prevent an array or object from being evaluated as an expression. This is necessary when an array or object needs to be returned by an expression.
<pre>['image', string]</pre>	string	Checks to see if a specified image ID is loaded into the maps image sprite. If it is, the ID is returned, otherwise null is returned.

EXPRESSION	RETURN TYPE	DESCRIPTION
['number', value] ["number", value, fallback: value, fallback: value, ...]	number	Asserts that the input value is a number. If multiple values are provided, each one is evaluated in order until a number is obtained. If none of the inputs are numbers, the expression is an error.
['object', value] ["object", value, fallback: value, fallback: value, ...]	Object	Asserts that the input value is an object. If multiple values are provided, each one is evaluated in order until an object is obtained. If none of the inputs are objects, the expression is an error.
['string', value] ["string", value, fallback: value, fallback: value, ...]	string	Asserts that the input value is a string. If multiple values are provided, each one is evaluated in order until a string is obtained. If none of the inputs are strings, the expression is an error.
['to-boolean', value]	boolean	Converts the input value to a boolean. The result is <code>false</code> when the input is an empty string, <code>0</code> , <code>false</code> , <code>null</code> , or <code>NaN</code> ; otherwise its <code>true</code> .
['to-color', value] ['to-color', value1, value2...]	color	Converts the input value to a color. If multiple values are provided, each one is evaluated in order until the first successful conversion is obtained. If none of the inputs can be converted, the expression is an error.
['to-number', value] ['to-number', value1, value2, ...]	number	Converts the input value to a number, if possible. If the input is <code>null</code> or <code>false</code> , the result is 0. If the input is <code>true</code> , the result is 1. If the input is a string, it's converted to a number using the <code>ToNumber</code> string function of the ECMAScript Language Specification. If multiple values are provided, each one is evaluated in order until the first successful conversion is obtained. If none of the inputs can be converted, the expression is an error.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>['to-string', value]</code>	string	Converts the input value to a string. If the input is <code>null</code> , the result is <code>""</code> . If the input is a boolean, the result is <code>"true"</code> or <code>"false"</code> . If the input is a number, it's converted to a string using the <code>ToString</code> number function of the ECMAScript Language Specification. If the input is a color, it's converted to CSS RGBA color string <code>"rgba(r,g,b,a)"</code> . Otherwise, the input is converted to a string using the <code>JSON.stringify</code> function of the ECMAScript Language Specification.
<code>['typeof', value]</code>	string	Returns a string describing the type of the given value.

TIP

If an error message similar to

```
Expression name must be a string, but found number instead. If you wanted a literal array, use
["literal", [...]].
```

appears in the browser console, then it means that there is an expression somewhere in your code that has an array that doesn't have a string for its first value. If you want the expression to return an array, wrap the array with the `literal` expression. The following example sets the icon `offset` option of a symbol layer, which needs to be an array containing two numbers, by using a `match` expression to choose between two offset values based on the value of the `entityType` property of the point feature.

```
var layer = new atlas.layer.SymbolLayer(datasource, null, {
  iconOptions: {
    offset: [
      'match',
      //Get the entityType value.
      ['get', 'entityType'],
      //If the entity type is 'restaurant', return a different pixel offset.
      ['restaurant', ['literal', [0, -10]]],
      //Default to value.
      ['literal', [0, 0]]
    ]
  }
});
```

Color expressions

Color expressions make it easier to create and manipulate color values.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>['rgb', number, number, number]</code>	color	Creates a color value from <code>red</code> , <code>green</code> , and <code>blue</code> components that must range between <code>0</code> and <code>255</code> , and an alpha component of <code>1</code> . If any component is out of range, the expression is an error.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>['rgba', number, number, number, number]</code>	color	Creates a color value from <i>red</i> , <i>green</i> , <i>blue</i> components that must range between <code>0</code> and <code>255</code> , and an alpha component within a range of <code>0</code> and <code>1</code> . If any component is out of range, the expression is an error.
<code>['to-rgba']</code>	[number, number, number, number]	Returns a four-element array containing the input color's <i>red</i> , <i>green</i> , <i>blue</i> , and <i>alpha</i> components, in that order.

Example

The following example creates an RGB color value that has a *red* value of `255`, and *green* and *blue* values that are calculated by multiplying `2.5` by the value of the `temperature` property. As the temperature changes, the color will change to different shades of *red*.

```
var layer = new atlas.layer.BubbleLayer(datasource, null, {
  color: [
    'rgb', //Create a RGB color value.

    255, //Set red value to 255.

    ['*', 2.5, ['get', 'temperature']], //Multiple the temperature by 2.5 and set the green value.

    ['*', 2.5, ['get', 'temperature']] //Multiple the temperature by 2.5 and set the blue value.
  ]
});
```

String operator expressions

String operator expressions perform conversion operations on strings such as concatenating and converting the case.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>['concat', string, string, ...]</code>	string	Concatenates multiple strings together. Each value must be a string. Use the <code>to-string</code> type expression to convert other value types to string if needed.
<code>['downcase', string]</code>	string	Converts the specified string to lowercase.
<code>['is-supported-script', string] ['is-supported-script', Expression]</code>	boolean	Determines if the input string uses a character set supported by the current font stack. For example:

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>['resolved-locale', string]</code>	string	Returns the IETF language tag of the locale being used by the provided collator. This can be used to determine the default system locale, or to determine if a requested locale was successfully loaded.
<code>['uppercase', string]</code>	string	Converts the specified string to uppercase.

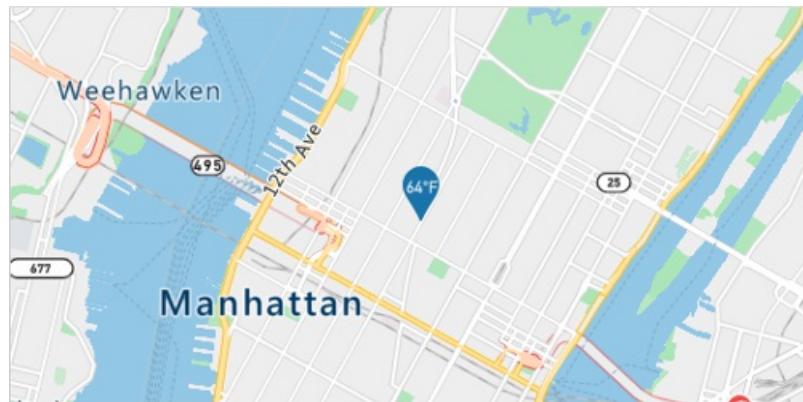
Example

The following example converts the `temperature` property of the point feature into a string and then concatenates "°F" to the end of it.

```
var layer = new atlas.layer.SymbolLayer(datasource, null, {
    textOptions: {
        textField: ['concat', ['to-string', ['get', 'temperature']], '°F'],

        //Some additional style options.
        offset: [0, -1.5],
        size: 12,
        color: 'white'
    }
});
```

The above expression renders a pin on the map with the text "64°F" overlaid on top of it as shown in the image below.



Interpolate and Step expressions

Interpolate and step expressions can be used to calculate values along an interpolated curve or step function.

These expressions take in an expression that returns a numeric value as their input, for example

`['get', 'temperature']`. The input value is evaluated against pairs of input and output values, to determine the value that best fits the interpolated curve or step function. The output values are called "stops". The input values for each stop must be a number and be in ascending order. The output values must be a number, and array of numbers, or a color.

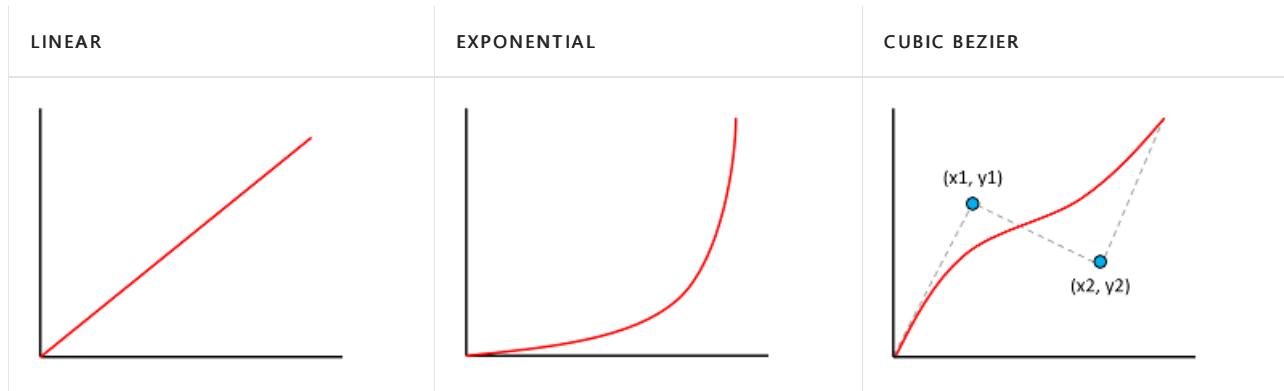
Interpolate expression

An `interpolate` expression can be used to calculate a continuous, smooth set of values by interpolating between stop values. An `interpolate` expression that returns color values produces a color gradient in which result values are selected from.

There are three types of interpolation methods that can be used in an `interpolate` expression:

- `['linear']` - Interpolates linearly between the pair of stops.
- `['exponential', base]` - Interpolates exponentially between the stops. The `base` value controls the rate at which the output increases. Higher values make the output increase more towards the high end of the range. A `base` value close to 1 produces an output that increases more linearly.
- `['cubic-bezier', x1, y1, x2, y2]` - Interpolates using a [cubic Bezier curve](#) defined by the given control points.

Here is an example of what these different types of interpolations look like.



The following pseudocode defines the structure of the `interpolate` expression.

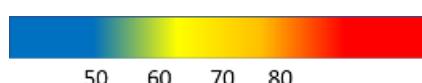
```
[  
  'interpolate',  
  interpolation: ['linear'] | ['exponential', base] | ['cubic-bezier', x1, y1, x2, y2],  
  input: number,  
  stopInput1: number,  
  stopOutput1: value1,  
  stopInput2: number,  
  stopOutput2: value2,  
  ...  
]
```

Example

The following example uses a `linear interpolate` expression to set the `color` property of a bubble layer based on the `temperature` property of the point feature. If the `temperature` value is less than 60, "blue" will be returned. If it's between 60 and less than 70, yellow will be returned. If it's between 70 and less than 80, "orange" will be returned. If it's 80 or greater, "red" will be returned.

```
var layer = new atlas.layer.BubbleLayer(datasource, null, {  
  color: [  
    'interpolate',  
    ['linear'],  
    ['get', 'temperature'],  
    50, 'blue',  
    60, 'yellow',  
    70, 'orange',  
    80, 'red'  
  ]  
});
```

The following image demonstrates how the colors are chosen for the above expression.



Step expression

A `step` expression can be used to calculate discrete, stepped result values by evaluating a [piecewise-constant function](#) defined by stops.

The following pseudocode defines the structure of the `step` expression.

```
[  
  'step',  
  input: number,  
  output0: value0,  
  stop1: number,  
  output1: value1,  
  stop2: number,  
  output2: value2,  
  ...  
]
```

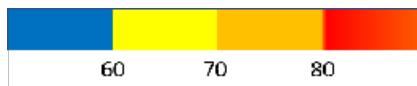
Step expressions return the output value of the stop just before the input value, or the first input value if the input is less than the first stop.

Example

The following example uses a `step` expression to set the `color` property of a bubble layer based on the `temperature` property of the point feature. If the `temperature` value is less than 60, "blue" will be returned. If it's between 60 and less than 70, "yellow" will be returned. If it's between 70 and less than 80, "orange" will be returned. If it's 80 or greater, "red" will be returned.

```
var layer = new atlas.layer.BubbleLayer(datasource, null, {  
  color: [  
    'step',  
    ['get', 'temperature'],  
    'blue',  
    60, 'yellow',  
    70, 'orange',  
    80, 'red'  
  ]  
});
```

The following image demonstrates how the colors are chosen for the above expression.



Layer-specific expressions

Special expressions that only apply to specific layers.

Heat map density expression

A heat map density expression retrieves the heat map density value for each pixel in a heat map layer and is defined as `['heatmap-density']`. This value is a number between `0` and `1`. It's used in combination with a `interpolation` or `step` expression to define the color gradient used to colorize the heat map. This expression can only be used in the `color` option of the heat map layer.

TIP

The color at index 0, in an interpolation expression or the default color of a step color, defines the color of the area where there's no data. The color at index 0 can be used to define a background color. Many prefer to set this value to transparent or a semi-transparent black.

Example

This example uses a liner interpolation expression to create a smooth color gradient for rendering the heat map.

```
var layer = new atlas.layer.HeatMapLayer(datasource, null, {
  color: [
    'interpolate',
    ['linear'],
    ['heatmap-density'],
    0, 'transparent',
    0.01, 'purple',
    0.5, '#fb00fb',
    1, '#00c3ff'
  ]
});
```

In addition to using a smooth gradient to colorize a heat map, colors can be specified within a set of ranges by using a `step` expression. Using a `step` expression for colorizing the heat map visually breaks up the density into ranges that resembles a contour or radar style map.

```
var layer = new atlas.layer.HeatMapLayer(datasource, null, {
  color: [
    'step',
    ['heatmap-density'],
    'transparent',
    0.01, 'navy',
    0.25, 'navy',
    0.5, 'green',
    0.75, 'yellow',
    1, 'red'
  ]
});
```

For more information, see the [Add a heat map layer](#) documentation.

Line progress expression

A line progress expression retrieves the progress along a gradient line in a line layer and is defined as `['line-progress']`. This value is a number between 0 and 1. It's used in combination with an `interpolation` or `step` expression. This expression can only be used with the [strokeGradient option](#) of the line layer.

NOTE

The `strokeGradient` option of the line layer requires the `lineMetrics` option of the data source to be set to `true`.

Example

This example uses the `['line-progress']` expression to apply a color gradient to the stroke of a line.

```

var layer = new atlas.layer.LineLayer(datasource, null, {
  strokeGradient: [
    'interpolate',
    ['linear'],
    ['line-progress'],
    0, "blue",
    0.1, "royalblue",
    0.3, "cyan",
    0.5, "lime",
    0.7, "yellow",
    1, "red"
  ]
});

```

[See live example](#)

Text field format expression

The text field format expression can be used with the `textField` option of the symbol layers `textOptions` property to provide mixed text formatting. This expression allows a set of input strings and formatting options to be specified. The following options can be specified for each input string in this expression.

- `'font-scale'` - Specifies the scaling factor for the font size. If specified, this value will override the `size` property of the `textOptions` for the individual string.
- `'text-font'` - Specifies one or more font families that should be used for this string. If specified, this value will override the `font` property of the `textOptions` for the individual string.

The following pseudocode defines the structure of the text field format expression.

```

[
  'format',
  input1: string,
  options1: {
    'font-scale': number,
    'text-font': string[]
  },
  input2: string,
  options2: {
    'font-scale': number,
    'text-font': string[]
  },
  ...
]

```

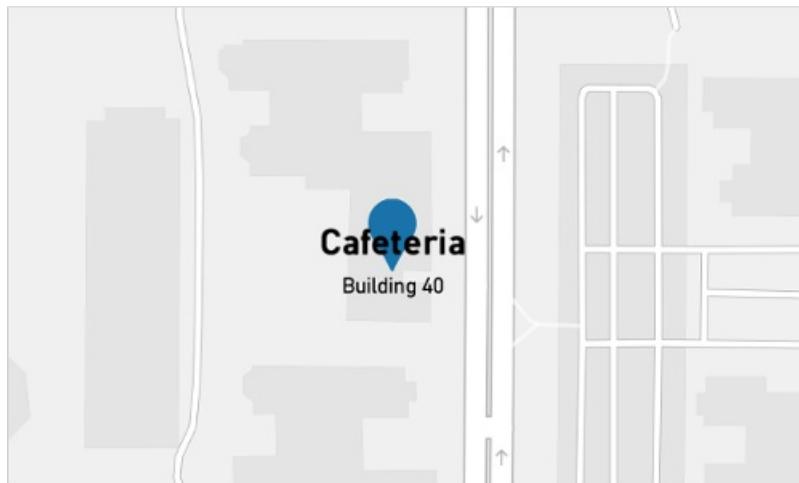
Example

The following example formats the text field by adding a bold font and scaling up the font size of the `title` property of the feature. This example also adds the `subTitle` property of the feature on a newline, with a scaled down font size.

```

var layer = new atlas.layer.SymbolLayer(datasource, null, {
  textOptions: [
    textField: [
      'format',
      //Bold the title property and scale its font size up.
      ['get', 'title'],
      {
        'text-font': ['literal', ['StandardFont-Bold']],
        'font-scale': 1.25
      },
      '\n', {}, //Add a new line without any formatting.
      //Scale the font size down of the subTitle property.
      ['get', 'subTitle'],
      {
        'font-scale': 0.75
      }
    ]
  }
});
```

This layer will render the point feature as shown in the image below:



Number format expression

The `number-format` expression can only be used with the `textField` option of a symbol layer. This expression converts the provided number into a formatted string. This expression wraps JavaScript's [Number.toLocaleString](#) function and supports the following set of options.

- `locale` - Specify this option for converting numbers to strings in a way that aligns with the specified language. Pass a [BCP 47 language tag](#) into this option.
- `currency` - To convert the number into a string representing a currency. Possible values are the [ISO 4217 currency codes](#), such as "USD" for the US dollar, "EUR" for the euro, or "CNY" for the Chinese RMB.
- `'min-fraction-digits'` - Specifies the minimum number of decimal places to include in the string version of the number.
- `'max-fraction-digits'` - Specifies the maximum number of decimal places to include in the string version of the number.

The following pseudocode defines the structure of the text field format expression.

```
[  
  'number-format',  
  input: number,  
  options: {  
    locale: string,  
    currency: string,  
    'min-fraction-digits': number,  
    'max-fraction-digits': number  
  }  
]
```

Example

The following example uses a `number-format` expression to modify how the `revenue` property of the point feature is rendered in the `textField` option of a symbol layer such that it appears a US dollar value.

```
var layer = new atlas.layer.SymbolLayer(datasource, null, {  
  textOptions: {  
    textField: [  
      'number-format',  
      ['get', 'revenue'],  
      { 'currency': 'USD' }  
    ],  
  
    offset: [0, 0.75]  
  }  
});
```

This layer will render the point feature as shown in the image below:

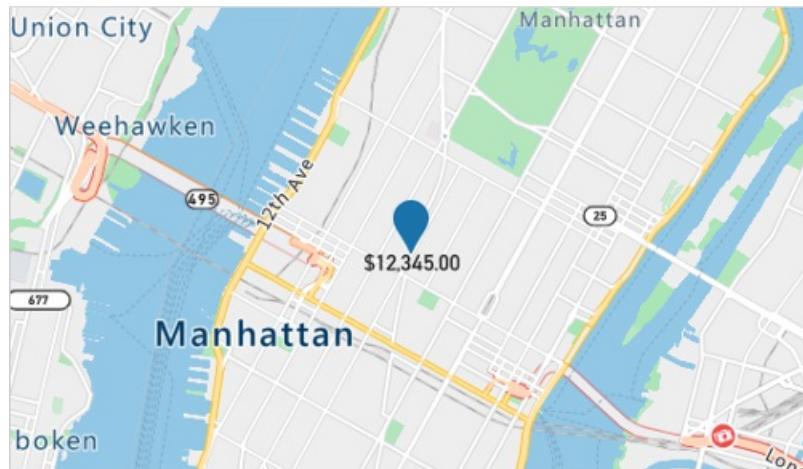


Image expression

An image expression can be used with the `image` and `textField` options of a symbol layer, and the `fillPattern` option of the polygon layer. This expression checks that the requested image exists in the style and will return either the resolved image name or `null`, depending on whether or not the image is currently in the style. This validation process is synchronous and requires the image to have been added to the style before requesting it in the image argument.

Example

The following example uses an `image` expression to add an icon inline with text in a symbol layer.

```

//Load the custom image icon into the map resources.
map.imageSprite.add('wifi-icon', 'wifi.png').then(function () {

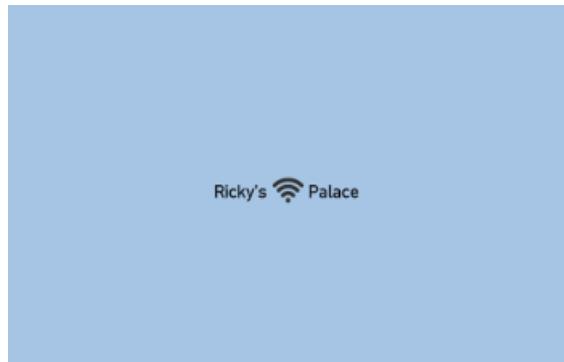
    //Create a data source and add it to the map.
    datasource = new atlas.source.DataSource();
    map.sources.add(datasource);

    //Create a point feature and add it to the data source.
    datasource.add(new atlas.data.Point(map.getCamera().center));

    //Add a layer for rendering point data as symbols.
    map.layers.add(new atlas.layer.SymbolLayer(datasource, null, {
        iconOptions: {
            image: 'none'
        },
        textOptions: {
            //Create a formatted text string that has an icon in it.
            textField: ["format", 'Ricky\'s ', ["image", "wifi-icon"], ' Palace']
        }
    }));
});

```

This layer will render the text field in the symbol layer as shown in the image below:



Zoom expression

A `zoom` expression is used to retrieve the current zoom level of the map at render time and is defined as `['zoom']`. This expression returns a number between the minimum and maximum zoom level range of the map. The Azure Maps interactive map controls for web and Android support 25 zoom levels, numbered 0 through 24. Using the `zoom` expression allows styles to be modified dynamically as the zoom level of the map is changed. The `zoom` expression may only be used with `interpolate` and `step` expressions.

Example

By default, the radii of data points rendered in the heat map layer have a fixed pixel radius for all zoom levels. As the map is zoomed, the data aggregates together and the heat map layer looks different. A `zoom` expression can be used to scale the radius for each zoom level such that each data point covers the same physical area of the map. It will make the heat map layer look more static and consistent. Each zoom level of the map has twice as many pixels vertically and horizontally as the previous zoom level. Scaling the radius, such that it doubles with each zoom level, will create a heat map that looks consistent on all zoom levels. It can be accomplished using the `zoom` expression with a `base 2 exponential interpolation` expression, with the pixel radius set for the minimum zoom level and a scaled radius for the maximum zoom level calculated as `2 * Math.pow(2, minZoom - maxZoom)` as shown below.

```

var layer = new atlas.layer.HeatMapLayer(datasource, null, {
  radius: [
    'interpolate',
    ['exponential', 2],
    ['zoom'],

    //For zoom level 1 set the radius to 2 pixels.
    1, 2,

    //Between zoom level 1 and 19, exponentially scale the radius from 2 pixels to 2 * Math.pow(2, 19 - 1) pixels (524,288 pixels).
    19, 2 * Math.pow(2, 19 - 1)
  ]
};


```

[See live example](#)

Variable binding expressions

Variable binding expressions store the results of a calculation in a variable. So, that the calculation results can be referenced elsewhere in an expression multiple times. It is a useful optimization for expressions that involve many calculations.

EXPRESSION	RETURN TYPE	DESCRIPTION
['let', name1: string, value1: any, name2: string, value2: any, ... childExpression]		Stores one or more values as variables for use by the <code>var</code> expression in the child expression that returns the result.
<code>['var', name: string]</code>	any	References a variable that was created using the <code>let</code> expression.

Example

This example uses an expression that calculates the revenue relative to temperature ratio and then uses a `case` expression to evaluate different boolean operations on this value. The `let` expression is used to store the revenue relative to temperature ratio, so that it only needs to be calculated once. The `var` expression references this variable as often as needed without having to recalculate it.

```
var layer = new atlas.layer.BubbleLayer(datasource, null, {
  color: [
    //Divide the point features `revenue` property by the `temperature` property and store it in a
    variable called `ratio`.
    'let', 'ratio', ['/[, ['get', 'revenue'], ['get', 'temperature']]],
    //Evaluate the child expression in which the stored variable will be used.
    [
      'case',
      //Check to see if the ratio is less than 100, return 'red'.
      ['<', ['var', 'ratio'], 100],
      'red',
      //Check to see if the ratio is less than 200, return 'green'.
      ['<', ['var', 'ratio'], 200],
      'green',
      //Return `blue` for values greater or equal to 200.
      'blue'
    ]
  ]
});
```

Next steps

See the following articles for more code samples that implement expressions:

[Add a symbol layer](#)

[Add a bubble layer](#)

[Add a line layer](#)

[Add a polygon layer](#)

[Add a heat map layer](#)

Learn more about the layer options that support expressions:

[BubbleLayerOptions](#)

[HeatMapLayerOptions](#)

[LineLayerOptions](#)

[PolygonLayerOptions](#)

[SymbolLayerOptions](#)

How to use image templates

11/2/2020 • 6 minutes to read • [Edit Online](#)

Images can be used with HTML markers and various layers within the Azure Maps web SDK:

- Symbol layers can render points on the map with an image icon. Symbols can also be rendered along a lines path.
- Polygon layers can be rendered with a fill pattern image.
- HTML markers can render points using images and other HTML elements.

In order to ensure good performance with layers, load the images into the map image sprite resource before rendering. The [IconOptions](#), of the [SymbolLayer](#), preloads a couple of marker images in a handful of colors into the map image sprite, by default. These marker images and more are available as SVG templates. They can be used to create images with custom scales, or used as a customer primary and secondary color. In total there are 42 image templates provided: 27 symbol icons and 15 polygon fill patterns.

Image templates can be added to the map image sprite resources by using the

`map.imageSprite.createFromTemplate` function. This function allows up to five parameters to be passed in;

```
createFromTemplate(id: string, templateName: string, color?: string, secondaryColor?: string, scale?: number): Promise<void>
```

The `id` is a unique identifier you create. The `id` is assigned to the image when it's added to the maps image sprite. Use this identifier in the layers to specifying which image resource to render. The `templateName` specifies which image template to use. The `color` option sets the primary color of the image and the `secondaryColor` options sets the secondary color of the image. The `scale` option scales the image template before applying it to the image sprite. When the image is applied to the image sprite, it's converted into a PNG. To ensure crisp rendering, it's better to scale up the image template before adding it to the sprite, than to scale it up in a layer.

This function asynchronously loads the image into the image sprite. Thus, it returns a Promise that you can wait for this function to complete.

The following code shows how to create an image from one of the built-in templates, and use it with a symbol layer.

```
map.imageSprite.createFromTemplate('myTemplatedIcon', 'marker-flat', 'teal', '#fff').then(function () {  
  
    //Add a symbol layer that uses the custom created icon.  
    map.layers.add(new atlas.layer.SymbolLayer(datasource, null, {  
        iconOptions: {  
            image: 'myTemplatedIcon'  
        }  
    }));  
});
```

Use an image template with a symbol layer

Once an image template is loaded into the map image sprite, it can be rendered as a symbol in a symbol layer by referencing the image resource ID in the `image` option of the `iconOptions`.

The following sample renders a symbol layer using the `marker-flat` image template with a teal primary color

and a white secondary color.

<https://codepen.io/azuremaps/embed/VoQMPp/%3Fheight=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Use an image template along a lines path

Once an image template is loaded into the map image sprite, it can be rendered along the path of a line by adding a LineString to a data source and using a symbol layer with a `lineSpacing` option and by referencing the ID of the image resource in the `image` option of the `iconOptions`.

The following sample renders a pink line on the map and uses a symbol layer using the `car` image template with a dodger blue primary color and a white secondary color.

<https://codepen.io/azuremaps/embed/KOQvJe/%3Fheight=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

TIP

If the image template points up, set the `rotation` icon option of the symbol layer to 90 if you want it to point in the same direction as the line.

Use an image template with a polygon layer

Once an image template is loaded into the map image sprite, it can be rendered as a fill pattern in a polygon layer by referencing the image resource ID in the `fillPattern` option of the layer.

The following sample renders a polygon layer using the `dot` image template with a red primary color and a transparent secondary color.

<https://codepen.io/azuremaps/embed/WVMErz/%3Fheight=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

TIP

Setting the secondary color of fill patterns makes it easier to see the underlying map will still providing the primary pattern.

Use an image template with an HTML marker

An image template can be retrieved using the `alta.getTemplate` function and used as the content of an HTML marker. The template can be passed into the `htmlContent` option of the marker, and then customized using the `color`, `secondaryColor`, and `text` options.

The following sample uses the `marker-arrow` template with a red primary color, a pink secondary color, and a text value of "00".

<https://codepen.io/azuremaps/embed/EqQvzq/%3Fheight=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

TIP

Image templates can be used outside of the map too. The `getImageTemplate` function returns an SVG string that has placeholders; `{color}`, `{secondaryColor}`, `{scale}`, `{text}`. Replace these placeholder values to create a valid SVG string. You can then either add the SVG string directly to the HTML DOM or convert it into a data URI and insert it into an image tag. For example:

```
//Retrieve an SVG template and replace the placeholder values.  
var svg = atlas.getImageTemplate('marker').replace(/{color}/, 'red').replace(/{secondaryColor}/,  
'white').replace(/{text}/, '').replace(/{scale}/, 1);  
  
//Convert to data URI for use in image tags.  
var dataUri = 'data:image/svg+xml;base64,' + btoa(svg);
```

Create custom reusable templates

If your application uses the same icon with different icons or if you are creating a module that adds additional image templates, you can easily add and retrieve these icons from the Azure Maps web SDK. Use the following static functions on the `atlas` namespace.

NAME	RETURN TYPE	DESCRIPTION
<code>addImageTemplate(templateName: string, template: string, override: boolean)</code>		Adds a custom SVG image template to the <code>atlas</code> namespace.
<code>getImageTemplate(templateName: string, scale?: number)</code>	string	Retrieves an SVG template by name.
<code>getAllImageTemplateNames()</code>	string[]	Retrieves an SVG template by name.

SVG image templates support the following placeholder values:

PLACEHOLDER	DESCRIPTION
<code>{color}</code>	The primary color.
<code>{secondaryColor}</code>	The secondary color.
<code>{scale}</code>	The SVG image is converted to an png image when added to the map image sprite. This placeholder can be used to scale a template before it is converted to ensure it renders clearly.
<code>{text}</code>	The location to render text when used with an HTML Marker.

The following example shows how to take an SVG template, and add it to the Azure Maps web SDK as a reusable icon template.

<https://codepen.io/azuremaps/embed/NQyvEX/?height=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

List of image templates

This table lists all image templates currently available within the Azure Maps web SDK. The template name is above each image. By default, the primary color is blue and the secondary color is white. To make the secondary color easier to see on a white background, the following images have the secondary color set to black.

Symbol icon templates

marker

marker-thick

marker-circle

marker-flat



marker-square

marker-square-cluster

marker-arrow

marker-ball-pin



marker-square-rounded

marker-square-rounded-cluster

flag

flag-triangle





triangle

triangle-thick

triangle-arrow-up

triangle-arrow-left



hexagon

hexagon-thick

hexagon-rounded

hexagon-rounded-thick



pin

pin-round

rounded-square

rounded-square-thick





arrow-up

arrow-up-thin

car



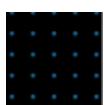
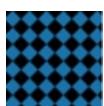
Polygon fill pattern templates

checker

checker-rotated

circles

circles-spaced

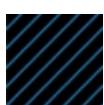


diagonal-lines-up

diagonal-lines-down

diagonal-stripes-up

diagonal-stripes-down





grid-lines

rotated-grid-lines

rotated-grid-stripes

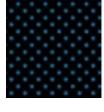
x-fill



zig-zag

zig-zag-vertical

dots



Preloaded image icons

The map preloads a set of icons into the maps image sprite using the `marker`, `pin`, and `pin-round` templates. These icon names and their color values are listed in the table below.

ICON NAME	COLOR	SECONDARY COLOR
<code>marker-black</code>	#231f20	#ffffff
<code>marker-blue</code>	#1a73aa	#ffffff
<code>marker-darkblue</code>	#003963	#ffffff
<code>marker-red</code>	#ef4c4c	#ffffff
<code>marker-yellow</code>	#f2c851	#ffffff
<code>pin-blue</code>	#2072b8	#ffffff
<code>pin-darkblue</code>	#003963	#ffffff
<code>pin-red</code>	#ef4c4c	#ffffff
<code>pin-round-blue</code>	#2072b8	#ffffff
<code>pin-round-darkblue</code>	#003963	#ffffff
<code>pin-round-red</code>	#ef4c4c	#ffffff

Try it now tool

With the following tool, you can render the different built-in image templates in various ways and customize the primary and secondary colors and scale.

<https://codepen.io/azuremaps/embed/NQyaaO/?height=500&theme-id=0&default-tab=result&rerun-position=hidden&>

Next steps

Learn more about the classes and methods used in this article:

[ImageSpriteManager](#)

[atlas namespace](#)

See the following articles for more code samples where image templates can be used:

[Add a symbol layer](#)

[Add a line layer](#)

[Add a polygon layer](#)

[Add HTML Makers](#)

Interact with the map

11/2/2020 • 5 minutes to read • [Edit Online](#)

This article shows you how to use [map events class](#). The property highlight events on the map and on different layers of the map. You can also highlight events when you interact with an HTML marker.

Interact with the map

Play with the map below, and see the corresponding mouse events highlighted on the right. You can click on the **JS tab** to view and edit the JavaScript code. You can also click on **Edit on CodePen** to modify the code on CodePen.

<https://codepen.io/azuremaps/embed/bLZEWd/?height=600&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Interact with map layers

The following code highlights the fired event as you interact with the Symbol Layer. The symbol, bubble, line, and polygon layer all support the same set of events. The heat map and tile layers don't support any of these events.

<https://codepen.io/azuremaps/embed/bQRRPE/?height=600&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Interact with HTML Marker

The following code adds JavaScript map events to an HTML marker. It also highlights the name of the events that get fired up as you interact with the HTML marker.

<https://codepen.io/azuremaps/embed/VVzKJY/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

The following table lists all of the supported map class events.

EVENT	DESCRIPTION
<code>boxzoomend</code>	Fired when a "box zoom" interaction ends.
<code>boxzoomstart</code>	Fired when a "box zoom" interaction starts.
<code>click</code>	Fired when a pointing device is pressed and released at the same point on the map.
<code>close</code>	Fired when the popup is closed manually or programatically.
<code>contextmenu</code>	Fired when the right button of the mouse is clicked.
<code>data</code>	Fired when any map data loads or changes.

EVENT	DESCRIPTION
<code>dataadded</code>	Fired when shapes are added to the <code>DataSource</code> .
<code>dataremoved</code>	Fired when shapes are removed from the <code>DataSource</code> .
<code>datasourceupdated</code>	Fired when the <code>DataSource</code> object is updated.
<code>dblclick</code>	Fired when a pointing device is clicked twice at the same point on the map.
<code>drag</code>	Fired repeatedly during a "drag to pan" interaction on the map, popup, or HTML marker.
<code>dragend</code>	Fired when a "drag to pan" interaction ends on the map, popup, or HTML marker.
<code>dragstart</code>	Fired when a "drag to pan" interaction starts on the map, popup, or HTML marker.
<code>error</code>	Fired when an error occurs.
<code>idle</code>	<p>Fired after the last frame rendered before the map enters an "idle" state:</p> <ul style="list-style-type: none"> • No camera transitions are in progress. • All currently requested tiles have loaded. • All fade/transition animations have completed.
<code>keydown</code>	Fired when a key is pressed down.
<code>keypress</code>	Fired when a key that produces a typable character (an ANSI key) is pressed.
<code>keyup</code>	Fired when a key is released.
<code>layeradded</code>	Fired when a layer is added to the map.
<code>layerremoved</code>	Fired when a layer is removed from the map.
<code>load</code>	Fired immediately after all necessary resources have been downloaded and the first visually complete rendering of the map has occurred.
<code>mousedown</code>	Fired when a pointing device is pressed within the map or when on top of an element.
<code>mouseenter</code>	Fired when a pointing device is initially moved over the map or an element.
<code>mouseleave</code>	Fired when a pointing device is moved out the map or an element.

EVENT	DESCRIPTION
<code>mousemove</code>	Fired when a pointing device is moved within the map or an element.
<code>mouseout</code>	Fired when a point device leaves the map's canvas our leaves an element.
<code>mouseover</code>	Fired when a pointing device is moved over the map or an element.
<code>mouseup</code>	Fired when a pointing device is released within the map or when on top of an element.
<code>move</code>	Fired repeatedly during an animated transition from one view to another, as the result of either user interaction or methods.
<code>moveend</code>	Fired just after the map completes a transition from one view to another, as the result of either user interaction or methods.
<code>movestart</code>	Fired just before the map begins a transition from one view to another, as the result of either user interaction or methods.
<code>open</code>	Fired when the popup is opened manually or programmatically.
<code>pitch</code>	Fired whenever the map's pitch (tilt) changes as the result of either user interaction or methods.
<code>pitchend</code>	Fired immediately after the map's pitch (tilt) finishes changing as the result of either user interaction or methods.
<code>pitchstart</code>	Fired whenever the map's pitch (tilt) begins a change as the result of either user interaction or methods.
<code>ready</code>	Fired when the minimum required map resources are loaded before the map is ready to be programmatically interacted with.
<code>render</code>	Fired whenever the map is drawn to the screen, as the result of: <ul style="list-style-type: none"> • A change to the map's position, zoom, pitch, or bearing. • A change to the map's style. • A change to a <code>DataSource</code> source. • The loading of a vector tile, GeoJSON file, glyph, or sprite.
<code>resize</code>	Fired immediately after the map has been resized.
<code>rotate</code>	Fired repeatedly during a "drag to rotate" interaction.

EVENT	DESCRIPTION
<code>rotateend</code>	Fired when a "drag to rotate" interaction ends.
<code>rotatestart</code>	Fired when a "drag to rotate" interaction starts.
<code>shapechanged</code>	Fired when a shape object property is changed.
<code>sourcedata</code>	Fired when one of the map's sources loads or changes, including if a tile belonging to a source loads or changes.
<code>sourceadded</code>	Fired when a <code>DataSource</code> or <code>VectorTileSource</code> is added to the map.
<code>sourceremoved</code>	Fired when a <code>DataSource</code> or <code>VectorTileSource</code> is removed from the map.
<code>styledata</code>	Fired when the map's style loads or changes.
<code>styleimagemissing</code>	Fired when a layer tries to load an image from the image sprite that doesn't exist
<code>tokenacquired</code>	Fired when an Azure Active Directory access token is obtained.
<code>touchcancel</code>	Fired when a <code>touchcancel</code> event occurs within the map.
<code>touchend</code>	Fired when a <code>touchend</code> event occurs within the map.
<code>touchmove</code>	Fired when a <code>touchmove</code> event occurs within the map.
<code>touchstart</code>	Fired when a <code>touchstart</code> event occurs within the map.
<code>wheel</code>	Fired when a mouse wheel event occurs within the map.
<code>zoom</code>	Fired repeatedly during an animated transition from one zoom level to another, as the result of either user interaction or methods.
<code>zoomend</code>	Fired just after the map completes a transition from one zoom level to another, as the result of either user interaction or methods.
<code>zoomstart</code>	Fired just before the map begins a transition from one zoom level to another, as the result of either user interaction or methods.

Next steps

See the following articles for full code examples:

[Using the Azure Maps Services module](#)

[Code samples](#)

Building an accessible application

4/28/2021 • 9 minutes to read • [Edit Online](#)

Upwards of 20% of internet users have a need for accessible web applications. As such, it's important to make sure your application is designed such that any user can easily use it. Rather than thinking of accessibility as a set of tasks to complete, think of it as part of your overall user experience. The more accessible your application, the more people who can use it.

When it comes to rich interactive content like a map, some common accessibility considerations are:

- Support the screen reader for users who have difficulty seeing the web application.
- Have multiple methods for interacting with and navigating the web application such as mouse, touch, and keyboard.
- Ensure color contrast is such that colors don't blend together and become hard to distinguish from each other.

The Azure Maps Web SDK comes prebuilt with many accessibility features such as:

- Screen reader descriptions when the map moves and when the user focuses on a control or popup.
- Mouse, touch, and keyboard support.
- Accessible color contrast support in the road map style.
- High contrast support.

Full accessibility conformance details for all Microsoft products can be found [here](#). Search for "Azure Maps web" to find the document specifically for the Azure Maps Web SDK.

Navigating the map

There are several different ways in which the map can be zoomed, panned, rotated, and pitched. The following details all the different ways to navigate the map.

Zoom the map

- Using a mouse, double-click the map to zoom in one level.
- Using a mouse, scroll the wheel to zoom the map.
- Using a touch screen, touch the map with two fingers and pinch together to zoom out or spread the fingers apart to zoom in.
- Using a touch screen, double tap the map to zoom in one level.
- With the map focused, use the Plus sign (+) or Equals sign (=) to zoom in one level.
- With the map focused, use the Minus sign, Hyphen (-), or Underscore (_) to zoom out one level.
- Using the zoom control with a mouse, touch or keyboard tab/enter keys.
- Press and hold the `Shift` button and press the left mouse button down on the map and drag to draw out an area to zoom the map into.
- Using some multi-touch pads, dragging two fingers up to zoom out, or down to zoom in.

Pan the map

- Using a mouse, press down with the left mouse button on the map and drag in any direction.
- Using a touch screen, touch the map and drag in any direction.
- With the map focused, use the arrow keys to move the map.

Rotate the map

- Using a mouse, press down with the right mouse button on the map and drag left or right.
- Using a touch screen, touch the map with two fingers and rotate.
- With the map focused, use the shift key and the left or right arrow keys.
- Using the rotation control with a mouse, touch or keyboard tab/enter keys.

Pitch the map

- Using the mouse, press down with the right mouse button on the map and drag up or down.
- Using a touch screen, touch the map with two fingers and drag them up or down together.
- With the map focused, use the shift key plus the up or down arrow keys.
- Using the pitch control with a mouse, touch or keyboard tab/enter keys.

Change the Map Style

Not all developers want all possible map styles to be available in their application. If the developer displays the style picker control of the map, then the user may change the map style using the mouse, a touch, or the keyboard with the tab or enter key. The developer can specify which map styles they want to make available in the map style picker control. Also, the developer can programmatically set and change the map style.

Use high contrast

- When the map control is loaded, it checks to see if high contrast is enabled and the browser supports it.
- The map control does not monitor the high contrast mode of the device. If the device mode changes, the map will not. Thus, the user will need to reload the map by refreshing the page.
- When high contrast is detected the map style will automatically switch to high contrast, and all built-in controls will use a high contrast style. For example, ZoomControl, PitchControl, CompassControl, StyleControl, and other built-in controls, will use a high contrast style.
- There are two type of high contrast, light and dark. If the type of high contrast can be detected by the map controls, then the behavior of the map will adjust accordingly. If light, then the `grayscale_light` map style will be loaded. If the type can't be detected or is dark, then the `high_contrast_dark` style will be loaded.
- If creating custom controls, it's useful to know if the built in controls are using a high contrast style. Developers can add a css class on the map container div to check. The css classes that would be added are `high-contrast-dark` and `high-contrast-light`. To check using JavaScript, use:

```
map.getMapContainer().classList.contains("high-contrast-dark")
```

or, use:

```
map.getMapContainer().classList.contains("high-contrast-light")
```

Keyboard shortcuts

The map has a number of keyboard shortcuts built in that make it easier to use the map. These keyboard shortcuts work when the map has focus.

KEY	ACTION
<code>Tab</code>	Navigate across the controls and popups in the map.

KEY	ACTION
<code>ESC</code>	Move focus from any element in the map to the top-level map element.
<code>Ctrl + Shift + D</code>	Toggle screen reader detail level.
Left arrow key	Pan the map left 100 pixels
Right arrow key	Pan the map right 100 pixels
Down arrow key	Pan the map down 100 pixels
Up arrow key	Pan the map up 100 pixels
<code>Shift + up arrow</code>	Increase map pitch by 10 degrees
<code>Shift + down arrow</code>	Decrease map pitch by 10 degrees
<code>Shift + right arrow</code>	Rotate the map 15 degrees clockwise
<code>Shift + left arrow</code>	Rotate the map 15 degrees counterclockwise
Plus sign (<code>+</code>) or *Equals sign (<code>=</code>)	Zoom in
Minus sign, Hyphen (<code>-</code>), or *Underscore (<code>_</code>)	Zoom out
<code>Shift + mouse drag on map to draw area</code>	Zoom into area

* These key shortcuts usually share the same key on a keyboard. These shortcuts were added to improve the user experience. It also doesn't matter if the user uses the shift key or not for these shortcuts.

Screen Reader support

Users can navigate the map using the keyboard. If a screen reader is running, the map will notify the user of changes to its state. For example, users are notified of map changes when the map is panned or zoomed. By default, the map provides simplified descriptions that exclude the zoom level and coordinates of the center of the map. The user can toggle the detail level of these descriptions by using the keyboard short cut `Ctrl + Shift + D`.

Any additional information that is placed on the base map should have corresponding textual information for screen reader users. Be sure to add [Accessible Rich Internet Applications \(ARIA\)](#), alt, and title attributes where appropriate.

Make popups keyboard accessible

A marker or symbol is often used to represent a location on the map. Additional information about the location is typically displayed in a popup when the user interacts with the marker. In most applications, popups appear when a user clicks or taps a marker. However, clicking and tapping require the user to use a mouse and a touch screen, respectively. A good practice is to make popups accessible when using a keyboard. This functionality can be achieved by creating a popup for each data point and adding it to the map.

The following example loads points of interests on the map using a symbol layer and adds a popup to the map

for each point of interest. A reference to each popup is stored in the properties of each data point. It can also be retrieved for a marker, such as when a marker is clicked. When focused on the map, pressing the tab key will allow the user to step through each popup on the map.

<https://codepen.io/azuremaps/embed/ZoVyzQ/?height=504&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Additional accessibility tips

Here are some additional tips to make your web-mapping application more accessible.

- If displaying many interactive point data on the map, consider reducing the clutter and use clustering.
- Ensure color contrast ratio between text/symbols and background colors is 4.5:1 or more.
- Keep your screen reader (ARIA, alt, and title attributes) messages short, descriptive, and meaningful. Avoid unnecessary jargon and acronyms.
- Try to optimize messages sent to the screen reader to provide short meaningful information that is easy for the user to digest. For example, if you want to update the screen reader at a high frequency, such as when the map is moving, consider doing the following points:
 - Wait until the map has finished moving to update the screen reader.
 - Throttle the updates to once every few seconds.
 - Combine messages together in a logical way.
- Avoid using color as the only means of conveying information. Use text, icons, or patterns to supplement or replace the color. Some considerations:
 - If using a bubble layer to show the relative value between data points, consider scaling the radius of each bubble, coloring the bubble, or both.
 - Consider using a symbol layer with different icons for different metric categories, such as triangles, stars, and squares. The symbol layer also supports scaling the size of the icon. A text label can also be displayed.
 - If displaying line data, the width can be used to represent weight or size. A dash-array pattern can be used to represent different categories of lines. A symbol layer can be used in combination with a line to overlay icons along the line. Using an arrow icon is useful for showing the flow or direction of the line.
 - If displaying polygon data, a pattern, such as stripes, can be used as an alternative to color.
- Some visualizations such as heatmaps, tile layers, and image layers aren't accessible for users with vision impairments. Some considerations:
 - Have the screen reader describe what the layer is displaying when added to the map. For example, if a weather radar tile layer is displayed, then have the screen reader say "Weather radar data is overlaid on the map."
- Limit the amount of functionality that requires a mouse hover. These functionalities will be inaccessible to users who are using a keyboard or touch device to interact with your application. Note, it's still a good practice to have a hover style for interactive content such as clickable icons, links, and buttons.
- Try navigating your application using the keyboard. Make sure tab ordering is logical.
- If creating keyboard shortcuts, try to limit it to two keys or less.

Next steps

Learn about accessibility in the Web SDK modules.

[Drawing tools accessibility](#)

Learn about developing accessible apps with Microsoft Learn:

Accessibility in Action Digital Badge Learning Path

Take a look at these useful accessibility tools:

[Developing accessible apps](#)

[WAI-ARIA Overview](#)

[Web Accessibility Evaluation Tool \(WAVE\)](#)

[WebAim color contrast checker](#)

[No Coffee Vision Simulator](#)

Use the drawing tools module

11/2/2020 • 2 minutes to read • [Edit Online](#)

The Azure Maps Web SDK provides a *drawing tools module*. This module makes it easy to draw and edit shapes on the map using an input device such as a mouse or touch screen. The core class of this module is the [drawing manager](#). The drawing manager provides all the capabilities needed to draw and edit shapes on the map. It can be used directly, and it's integrated with a custom toolbar UI. You can also use the built-in [drawing toolbar](#) class.

Loading the drawing tools module in a webpage

1. Create a new HTML file and [implement the map as usual](#).
2. Load the Azure Maps drawing tools module. You can load it in one of two ways:
 - Use the globally hosted, Azure Content Delivery Network version of the Azure Maps services module. Add reference to the JavaScript and CSS stylesheet in the `<head>` element of the file:

```
<link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/drawing/0/atlas-drawing.min.css" type="text/css" />
<script src="https://atlas.microsoft.com/sdk/javascript/drawing/0/atlas-drawing.min.js">
</script>
```

- Or, you can load the drawing tools module for the Azure Maps Web SDK source code locally by using the [azure-maps-drawing-tools](#) npm package, and then host it with your app. This package also includes TypeScript definitions. Use this command:

```
npm install azure-maps-drawing-tools
```

Then, add a reference to the JavaScript and CSS stylesheet in the `<head>` element of the file:

```
<link rel="stylesheet" href="node_modules/azure-maps-drawing-tools/dist/atlas-drawing.min.css" type="text/css" />
<script src="node_modules/azure-maps-drawing-tools/dist/atlas-drawing.min.js"></script>
```

Use the drawing manager directly

Once the drawing tools module is loaded in your application, you can enable drawing and editing capabilities using the [drawing manager](#). You can specify options for the drawing manager while instantiating it or alternatively use the `drawingManager.setOptions()` function.

Set the drawing mode

The following code creates an instance of the drawing manager and sets the drawing **mode** option.

```
//Create an instance of the drawing manager and set drawing mode.
drawingManager = new atlas.drawing.DrawingManager(map,{
    mode: "draw-polygon"
});
```

The code below is a complete running example of how to set a drawing mode of the drawing manager. Click the map to start drawing a polygon.

<https://codepen.io/azuremaps/embed/YzKVKRa/?height=265&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Set the interaction type

The drawing manager supports three different ways of interacting with the map to draw shapes.

- `click` - Coordinates are added when the mouse or touch is clicked.
- `freehand` - Coordinates are added when the mouse or touch is dragged on the map.
- `hybrid` - Coordinates are added when the mouse or touch is clicked or dragged.

The following code enables the polygon drawing mode and sets the type of drawing interaction that the drawing manager should adhere to `freehand`.

```
//Create an instance of the drawing manager and set drawing mode.  
drawingManager = new atlas.drawing.DrawingManager(map, {  
    mode: "draw-polygon",  
    interactionType: "freehand"  
});
```

This code sample implements the functionality of drawing a polygon on the map. Just hold down the left mouse button and dragging it around, freely.

<https://codepen.io/azuremaps/embed/ZEzKoaj/?height=265&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Customizing drawing options

The previous examples demonstrated how to customize drawing options while instantiating the Drawing Manager. You can also set the Drawing Manager options by using the `drawingManager.setOptions()` function. Below is a tool to test out customization of all options for the drawing manager using the `setOptions` function.

<https://codepen.io/azuremaps/embed/LYPyxrR/?height=600&theme-id=0&default-tab=result&rerun-position=hidden&>

Next steps

Learn how to use additional features of the drawing tools module:

[Add a drawing toolbar](#)

[Get shape data](#)

[React to drawing events](#)

[Interaction types and keyboard shortcuts](#)

Learn more about the classes and methods used in this article:

[Map](#)

[Drawing manager](#)

[Drawing toolbar](#)

Add a drawing tools toolbar to a map

11/2/2020 • 2 minutes to read • [Edit Online](#)

This article shows you how to use the Drawing Tools module and display the drawing toolbar on the map. The `DrawingToolbar` control adds the drawing toolbar on the map. You will learn how to create maps with only one and all drawing tools and how to customize the rendering of the drawing shapes in the drawing manager.

Add drawing toolbar

The following code creates an instance of the drawing manager and displays the toolbar on the map.

```
//Create an instance of the drawing manager and display the drawing toolbar.  
drawingManager = new atlas.drawing.DrawingManager(map, {  
    toolbar: new atlas.control.DrawingToolbar({  
        position: 'top-right',  
        style: 'dark'  
    })  
});
```

Below is the complete running code sample of the functionality above:

<https://codepen.io/azuremaps/embed/ZEzLeRg/?height=265&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Limit displayed toolbar options

The following code creates an instance of the drawing manager and displays the toolbar with just a polygon drawing tool on the map.

```
//Create an instance of the drawing manager and display the drawing toolbar with polygon drawing tool.  
drawingManager = new atlas.drawing.DrawingManager(map, {  
    toolbar: new atlas.control.DrawingToolbar({  
        position: 'top-right',  
        style: 'light',  
        buttons: ["draw-polygon"]  
    })  
});
```

Below is the complete running code sample of the functionality above:

<https://codepen.io/azuremaps/embed/OJLWWMy/?height=265&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Change drawing rendering style

The style of the shapes that are drawn can be customized by retrieving the underlying layers of the drawing manager by using the `drawingManager.getLayers()` function and then setting options on the individual layers. The drag handles that appear for coordinates when editing a shape are HTML markers. The style of the drag handles can be customized by passing HTML marker options into the `dragHandleStyle` and `secondaryDragHandleStyle` options of the drawing manager.

The following code gets the rendering layers from the drawing manager and modifies their options to change rendering style for drawing. In this case, points will be rendered with a blue marker icon. Lines will be red and four pixels wide. Polygons will have a green fill color and an orange outline. It then changes the styles of the drag handles to be square icons.

```
//Get rendering layers of drawing manager.
var layers = drawingManager.getLayers();

//Change the icon rendered for points.
layers.pointLayer.setOptions({
    iconOptions: {
        image: 'marker-blue'
    }
});

//Change the color and width of lines.
layers.lineLayer.setOptions({
    strokeColor: 'red',
    strokeWidth: 4
});

//Change fill color of polygons.
layers.polygonLayer.setOptions({
    fillColor: 'green'
});

//Change the color of polygon outlines.
layers.polygonOutlineLayer.setOptions({
    strokeColor: 'orange'
});

//Update the style of the drag handles that appear when editing.
drawingManager.setOptions({
    //Primary drag handle that represents coordinates in the shape.
    dragHandleStyle: {
        anchor: 'center',
        htmlContent: '<svg width="15" height="15" viewBox="0 0 15 15" xmlns="http://www.w3.org/2000/svg" style="cursor:pointer"><rect x="0" y="0" width="15" height="15" style="stroke:black;fill:white;stroke-width:4px;" /></svg>',
        draggable: true
    },
    //Secondary drag handle that represents mid-point coordinates that users can grab to add new coordinates in the middle of segments.
    secondaryDragHandleStyle: {
        anchor: 'center',
        htmlContent: '<svg width="10" height="10" viewBox="0 0 10 10" xmlns="http://www.w3.org/2000/svg" style="cursor:pointer"><rect x="0" y="0" width="10" height="10" style="stroke:white;fill:black;stroke-width:4px;" /></svg>',
        draggable: true
    }
});
```

Below is the complete running code sample of the functionality above:

<https://codepen.io/azuremaps/embed/OJLWpyj/?height=265&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Next steps

Learn how to use additional features of the drawing tools module:

[Get shape data](#)

[React to drawing events](#)

[Interaction types and keyboard shortcuts](#)

Learn more about the classes and methods used in this article:

[Map](#)

[Drawing toolbar](#)

[Drawing manager](#)

Get shape data

11/2/2020 • 2 minutes to read • [Edit Online](#)

This article shows you how to get data of shapes that are drawn on the map. We use the `drawingManager.getSource()` function inside [drawing manager](#). There are various scenarios when you want to extract geojson data of a drawn shape and use it elsewhere.

Get data from drawn shape

The following function gets the drawn shape's source data and outputs it to the screen.

```
function getDrawnShapes() {  
    var source = drawingManager.getSource();  
  
    document.getElementById('CodeOutput').value = JSON.stringify(source.toJson(), null, '    ');  
}
```

Below is the complete running code sample, where you can draw a shape to test the functionality:

<https://codepen.io/azuremaps/embed/xxKgBVz/?height=265&theme-id=0&default-tab=result&rerun-position=hidden&>

Next steps

Learn how to use additional features of the drawing tools module:

[React to drawing events](#)

[Interaction types and keyboard shortcuts](#)

Learn more about the classes and methods used in this article:

[Map](#)

[Drawing manager](#)

[Drawing toolbar](#)

Drawing tool events

11/2/2020 • 2 minutes to read • [Edit Online](#)

When using drawing tools on a map, it's useful to react to certain events as the user draws on the map. This table lists all of the events supported by the `DrawingManager` class.

EVENT	DESCRIPTION
<code>drawingchanged</code>	Fired when any coordinate in a shape has been added or changed.
<code>drawingchanging</code>	Fired when any preview coordinate for a shape is being displayed. For example, this event will fire multiple times as a coordinate is dragged.
<code>drawingcomplete</code>	Fired when a shape has finished being drawn or taken out of edit mode.
<code>drawingmodechanged</code>	Fired when the drawing mode has changed. The new drawing mode is passed into the event handler.
<code>drawingstarted</code>	Fired when the user starts drawing a shape or puts a shape into edit mode.

The following code shows how the events in the Drawing Tools module work. Draw shapes on the map and watch as the events fire.

<https://codepen.io/azuremaps/embed/dyPMRWo?height=500&theme-id=default&default-tab=js,result&editable=true&rerun-position=hidden&>

Examples

Let's see some common scenarios that use the drawing tools events.

Select points in polygon area

This code demonstrates how to monitor an event of a user drawing shapes. For this example, the code monitors shapes of polygons, rectangles, and circles. Then, it determines which data points on the map are within the drawn area. The `drawingcomplete` event is used to trigger the select logic. In the select logic, the code loops through all the data points on the map. It checks if there's an intersection of the point and the area of the drawn shape. This example makes use of the open-source [Turf.js](#) library to perform a spatial intersection calculation.

<https://codepen.io/azuremaps/embed/XWJdeja?height=500&theme-id=default&default-tab=result&rerun-position=hidden&>

Draw and search in polygon area

This code searches for points of interests inside the area of a shape after the user finished drawing the shape. You can modify and execute the code by clicking 'Edit on Code pen' on the top-right corner of the frame. The `drawingcomplete` event is used to trigger the search logic. If the user draws a rectangle or polygon, a search inside geometry is performed. If a circle is drawn, the radius and center position is used to perform a point of

interest search. The `drawingmodechanged` event is used to determine when the user switches to the drawing mode, and this event clears the drawing canvas.

<https://codepen.io/azuremaps/embed/eYmZGNv?height=500&theme-id=default&default-tab=js,result&editable=true&rerun-position=hidden&>

Create a measuring tool

The code below shows how the drawing events can be used to create a measuring tool. The `drawingchanging` is used to monitor the shape, as it's being drawn. As the user moves the mouse, the dimensions of the shape are calculated. The `drawingcomplete` event is used to do a final calculation on the shape after it has been drawn. The `drawingmodechanged` event is used to determine when the user is switching into a drawing mode. Also, the `drawingmodechanged` event clears the drawing canvas and clears old measurement information.

<https://codepen.io/azuremaps/embed/RwNaZXe?height=500&theme-id=default&default-tab=js,result&editable=true&rerun-position=hidden&>

Next steps

Learn how to use additional features of the drawing tools module:

[Get shape data](#)

[Interaction types and keyboard shortcuts](#)

Learn more about the Services module:

[Services module](#)

Check out more code samples:

[Code sample page](#)

Interaction types and keyboard shortcuts in the drawing tools module

11/2/2020 • 6 minutes to read • [Edit Online](#)

This article outlines all the different ways to draw and edit shapes using a mouse, touch screen, or keyboard shortcuts.

The drawing manager supports three different ways of interacting with the map, to draw shapes.

- `click` - Coordinates are added when the mouse or touch is clicked.
- `freehand` - Coordinates are added when the mouse or touch is dragged on the map.
- `hybrid` - Coordinates are added when the mouse or touch is clicked or dragged.

How to draw shapes

Before any shape can be drawn, set the `drawingMode` option of the drawing manager to a supported drawing setting. This setting can be programmed, or invoked by pressing one of the drawing buttons on the toolbar. The drawing mode stays enabled, even after a shape has been drawn, making it easy to draw additional shapes of the same type. Programmatically set the drawing mode to an idle state. Or, switch to an idle state by clicking the current drawing modes button on the toolbar.

The next sections outline all the different ways that shapes can be drawn on the map.

How to draw a point

When the drawing manager is in `draw-point` drawing mode, the following actions can be done to draw points on the map. These methods work with all interaction modes.

Start drawing

- Click the left mouse button, or touch the map to add a point to the map.
- If the mouse is over the map, press the `F` key, and a point will be added at the coordinate of the mouse pointer. This method provides higher accuracy for adding a point to the map. There will be less movement on the mouse due to the pressing motion of the left mouse button.
- Keep clicking, touching, or pressing `F` to add more points to the map.

Finish drawing

- Click on any button in the drawing toolbar.
- Programmatically set the drawing mode.
- Press the `C` key.

Cancel drawing

- Press the `Escape` key.

How to draw a line

When the drawing manager is in `draw-line` mode, the following actions can be done to draw points on the map, depending on the interaction mode.

Start drawing

- Click mode

- Click the left mouse button, or touch the map to add each point of a line on the map. A coordinate is added to the line for each click or touch.
- If the mouse is over the map, press the `F` key, and a point will be added at the coordinate of the mouse pointer. This method provides higher accuracy for adding a point to the map. There will be less movement on the mouse due to the pressing motion of the left mouse button.
- Keep clicking until all the desired points have been added to the line.
- Freehand mode
 - Press down the left mouse button, or touch-down on the map and drag the mouse, or touch point around. Coordinates are added to the line as the mouse or touch point moves around the map. As soon as the mouse or touch-up event is triggered, the drawing is completed. The frequency at which coordinates are added is defined by the drawing managers `freehandInterval` option.
- Hybrid mode
 - Alternate between click and freehand methods, as desired, while drawing a single line. For example, click a few points, then hold and drag the mouse to add a bunch of points, then click a few more.

Finish drawing

- Hybrid/Click mode
 - Double-click the map at the last point.
 - Click on any button in the drawing toolbar.
 - Programmatically set the drawing mode.
- Freehand mode
 - Release the mouse button or touch point.
- Press the `c` key.

Cancel drawing

- Press the `Escape` key.

How to draw a polygon

When the drawing manager is in `draw-polygon` mode, the following actions can be done to draw points on the map, depending on the interaction mode.

Start drawing

- Click mode
 - Click the left mouse button, or touch the map to add each point of a polygon on the map. A coordinate is added to the polygon for each click or touch.
 - If the mouse is over the map, press the `F` key, and a point will be added at the coordinate of the mouse pointer. This method provides higher accuracy for adding a point to the map. There will be less movement on the mouse due to the pressing motion of the left mouse button.
 - Keep clicking until all the desired points have been added to the polygon.
- Freehand mode
 - Press down the left mouse button, or touch-down on the map and drag the mouse, or touch point around. Coordinates are added to the polygon as the mouse or touch point moves around the map. As soon as the mouse or touch-up event is triggered, the drawing is completed. The frequency at which coordinates are added is defined by the drawing managers `freehandInterval` option.
- Hybrid mode
 - Alternate between click and freehand methods, as desired, while drawing a single polygon. For example, click a few points, then hold and drag the mouse to add a bunch of points, then click a few more.

Finish drawing

- Hybrid/Click mode
 - Double-click the map at the last point.
 - Click on the first point in the polygon.
 - Click on any button in the drawing toolbar.
 - Programmatically set the drawing mode.
- Freehand mode
 - Release the mouse button or touch point.
- Press the `c` key.

Cancel drawing

- Press the `Escape` key.

How to draw a rectangle

When the drawing manager is in `draw-rectangle` mode, the following actions can be done to draw points on the map, depending on the interaction mode. The generated shape will follow the [extended GeoJSON specification for rectangles](#).

Start drawing

- Press down the left mouse button, or touch-down on the map to add the first corner of the rectangle and drag to create the rectangle.

Finish drawing

- Release the mouse button or touch point.
- Programmatically set the drawing mode.
- Press the `c` key.

Cancel drawing

- Press the `Escape` key.

How to draw a circle

When the drawing manager is in `draw-circle` mode, the following actions can be done to draw points on the map, depending on the interaction mode. The generated shape will follow the [extended GeoJSON specification for circles](#).

Start drawing

- Press down the left mouse button, or touch-down on the map to add the center of the circle and drag give the circles a radius.

Finish drawing

- Release the mouse button or touch point.
- Programmatically set the drawing mode.
- Press the `c` key.

Cancel drawing

- Press the `Escape` key.

Keyboard shortcuts

The drawing tools support keyboard shortcuts. These keyboard shortcuts are functional when the map is in focus.

KEY	ACTION
C	Completes any drawing that is in progress and sets the drawing mode to idle. Focus will move to top-level map element.
Escape	Cancels any drawing that is in progress and sets the drawing mode to idle. Focus will move to top-level map element.
F	Adds a coordinate to a point, line, or polygon if the mouse is over the map. Equivalent action of clicking the map when in click or hybrid mode. This shortcut allows for more precise and faster drawings. You can use one hand to position the mouse and other to press the button without moving the mouse from the press gesture.

Next steps

Learn more about the classes in the drawing tools module:

[Drawing manager](#)

[Drawing toolbar](#)

Use the Azure Maps services module

11/2/2020 • 4 minutes to read • [Edit Online](#)

The Azure Maps Web SDK provides a *services module*. This module is a helper library that makes it easy to use the Azure Maps REST services in web or Node.js applications by using JavaScript or TypeScript.

Use the services module in a webpage

1. Create a new HTML file.
2. Load the Azure Maps services module. You can load it in one of two ways:

- Use the globally hosted, Azure Content Delivery Network version of the Azure Maps services module. Add a script reference to the `<head>` element of the file:

```
<script src="https://atlas.microsoft.com/sdk/javascript/service/2/atlas-service.min.js">
</script>
```

- Alternatively, load the services module for the Azure Maps Web SDK source code locally by using the `azure-maps-rest` npm package, and then host it with your app. This package also includes TypeScript definitions. Use this command:

```
npm install azure-maps-rest
```

Then, add a script reference to the `<head>` element of the file:

```
<script src="node_modules/azure-maps-rest/dist/atlas-service.min.js"></script>
```

3. Create an authentication pipeline. The pipeline must be created before you can initialize a service URL client endpoint. Use your own Azure Maps account key or Azure Active Directory (Azure AD) credentials to authenticate an Azure Maps Search service client. In this example, the Search service URL client will be created.

If you use a subscription key for authentication:

```
// Get an Azure Maps key at https://azure.com/maps.
var subscriptionKey = '<Your Azure Maps Key>';

// Use SubscriptionKeyCredential with a subscription key.
var subscriptionKeyCredential = new atlas.service.SubscriptionKeyCredential(subscriptionKey);

// Use subscriptionKeyCredential to create a pipeline.
var pipeline = atlas.service.MapsURL.newPipeline(subscriptionKeyCredential, {
    retryOptions: { maxTries: 4 } // Retry options
});

// Create an instance of the SearchURL client.
var searchURL = new atlas.service.SearchURL(pipeline);
```

If you use Azure AD for authentication:

```

// Enter your Azure AD client ID.
var clientId = "<Your Azure Active Directory Client Id>";

// Use TokenCredential with OAuth token (Azure AD or Anonymous).
var aadToken = await getAadToken();
var tokenCredential = new atlas.service.TokenCredential(clientId, aadToken);

// Create a repeating time-out that will renew the Azure AD token.
// This time-out must be cleared when the TokenCredential object is no longer needed.
// If the time-out is not cleared, the memory used by the TokenCredential will never be reclaimed.
var renewToken = async () => {
    try {
        console.log("Renewing token");
        var token = await getAadToken();
        tokenCredential.token = token;
        tokenRenewalTimer = setTimeout(renewToken, getExpiration(token));
    } catch (error) {
        console.log("Caught error when renewing token");
        clearTimeout(tokenRenewalTimer);
        throw error;
    }
}
tokenRenewalTimer = setTimeout(renewToken, getExpiration(aadToken));

// Use tokenCredential to create a pipeline.
var pipeline = atlas.service.MapsURL.newPipeline(tokenCredential, {
    retryOptions: { maxTries: 4 } // Retry options
});

// Create an instance of the SearchURL client.
var searchURL = new atlas.service.SearchURL(pipeline);

function getAadToken() {
    // Use the signed-in auth context to get a token.
    return new Promise((resolve, reject) => {
        // The resource should always be https://atlas.microsoft.com/.
        const resource = "https://atlas.microsoft.com/";
        authContext.acquireToken(resource, (error, token) => {
            if (error) {
                reject(error);
            } else {
                resolve(token);
            }
        });
    });
}

function getExpiration(jwtToken) {
    // Decode the JSON Web Token (JWT) to get the expiration time stamp.
    const json = atob(jwtToken.split(".")[1]);
    const decode = JSON.parse(json);

    // Return the milliseconds remaining until the token must be renewed.
    // Reduce the time until renewal by 5 minutes to avoid using an expired token.
    // The exp property is the time stamp of the expiration, in seconds.
    const renewSkew = 300000;
    return (1000 * decode.exp) - Date.now() - renewSkew;
}

```

For more information, see [Authentication with Azure Maps](#).

4. The following code uses the newly created Azure Maps Search service URL client to geocode an address: "1 Microsoft Way, Redmond, WA". The code uses the `searchAddress` function and displays the results as a table in the body of the page.

```

// Search for "1 microsoft way, redmond, wa".
searchURL.searchAddress(atlas.serviceAborter.timeout(10000), '1 microsoft way, redmond, wa')
.then(response => {
  var html = [];

  // Display the total results.
  html.push('Total results: ', response.summary.numResults, '<br/><br/>');

  // Create a table of the results.
  html.push('<table><tr><td></td><td>Result</td><td>Latitude</td><td>Longitude</td></tr>');

  for(var i=0;i<response.results.length;i++){
    html.push('<tr><td>', (i+1), '.</td><td>',
      response.results[i].address.freeformAddress,
      '</td><td>',
      response.results[i].position.lat,
      '</td><td>',
      response.results[i].position.lon,
      '</td></tr>');
  }

  html.push('</table>');

  // Add the resulting HTML to the body of the page.
  document.body.innerHTML = html.join('');
});

```

Here's the full, running code sample:

<https://codepen.io/azuremaps/embed/zbXGMR/?height=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

Azure Government cloud support

The Azure Maps Web SDK supports the Azure Government cloud. All JavaScript and CSS URLs used to access the Azure Maps Web SDK remain the same, however the following tasks will need to be done to connect to the Azure Government cloud version of the Azure Maps platform.

When using the interactive map control, add the following line of code before creating an instance of the `Map` class.

```
atlas.setDomain('atlas.azure.us');
```

Be sure to use an Azure Maps authentication details from the Azure Government cloud platform when authenticating the map and services.

When using the services module, the domain for the services needs to be set when creating an instance of an API URL endpoint. For example, the following code creates an instance of the `SearchURL` class and points the domain to the Azure Government cloud.

```
var searchURL = new atlas.service.SearchURL(pipeline, 'atlas.azure.us');
```

If directly accessing the Azure Maps REST services, change the URL domain to `atlas.azure.us`. For example, if using the search API service, change the URL domain from <https://atlas.microsoft.com/search/> to <https://atlas.azure.us/search/>.

Next steps

Learn more about the classes and methods used in this article:

[MapsURL](#)

[SearchURL](#)

[RouteURL](#)

[SubscriptionKeyCredential](#)

[TokenCredential](#)

For more code samples that use the services module, see these articles:

[Show search results on the map](#)

[Get information from a coordinate](#)

[Show directions from A to B](#)

Show search results on the map

11/2/2020 • 3 minutes to read • [Edit Online](#)

This article shows you how to search for location of interest and show the search results on the map.

There are two ways to search for a location of interest. One way is to use a service module to make a search request. The other way is to make a search request to [Azure Maps Fuzzy search API](#) through the [Fetch API](#). Both ways are discussed below.

Make a search request via service module

<https://codepen.io/azuremaps/embed/zLdYEB/%3Fheight=265&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

In the code above, the first block constructs a map object and sets the authentication mechanism to use the access token. You can see [create a map](#) for instructions.

The second block of code creates a `TokenCredential` to authenticate HTTP requests to Azure Maps with the access token. It then passes the `TokenCredential` to `atlas.service.MapsURL.newPipeline()` and creates a `Pipeline` instance. The `searchURL` represents a URL to Azure Maps [Search](#) operations.

The third block of code creates a data source object using the `DataSource` class and add search results to it. A [symbol layer](#) uses text or icons to render point-based data wrapped in the `DataSource` as symbols on the map. A symbol layer is then created. The data source is added to the symbol layer, which is then added to the map.

The fourth code block uses the `SearchFuzzy` method in the [service module](#). It allows you to perform a free form text search via the [Get Search Fuzzy rest API](#) to search for point of interest. Get requests to the Search Fuzzy API can handle any combination of fuzzy inputs. A GeoJSON feature collection from the response is then extracted using the `geojson.getFeatures()` method and added to the data source, which automatically results in the data being rendered on the map via the symbol layer.

The last block of code adjusts the camera bounds for the map using the Map's `setCamera` property.

The search request, data source, symbol layer, and camera bounds are inside the `event listener` of the map. We want to ensure that the results are displayed after the map fully loads.

Make a search request via Fetch API

<https://codepen.io/azuremaps/embed/KQbaeM/%3Fheight=265&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

In the code above, the first block of code constructs a map object. It sets the authentication mechanism to use the access token. You can see [create a map](#) for instructions.

The second block of code creates a URL to make a search request to. It also creates two arrays to store bounds and pins for search results.

The third block of code uses the [Fetch API](#). The [Fetch API](#) is used to make a request to [Azure Maps Fuzzy search API](#) to search for the points of interest. The Fuzzy search API can handle any combination of fuzzy inputs. It then handles and parses the search response and adds the result pins to the `searchPins` array.

The fourth block of code creates a data source object using the `DataSource` class. In the code, we add search results to the source object. A [symbol layer](#) uses text or icons to render point-based data wrapped in the `DataSource` as symbols on the map. A symbol layer is then created. The data source is added to the symbol layer,

which is then added to the map.

The last block of code creates a [BoundingBox](#) object. It uses the array of results, and then it adjusts the camera bounds for the map using the Map's [setCamera](#). It then renders the result pins.

The search request, the data source, symbol layer, and the camera bounds are set within the map's [event listener](#) to ensure that the results are displayed after the map loads fully.

Next steps

[Best practices for using the search service](#)

Learn more about **Fuzzy Search**:

[Azure Maps Fuzzy Search API](#)

Learn more about the classes and methods used in this article:

[Map](#)

See the following articles for full code examples:

[Get information from a coordinate](#)

[Show directions from A to B](#)

Get information from a coordinate

11/2/2020 • 2 minutes to read • [Edit Online](#)

This article shows how to make a reverse address search that shows the address of a clicked popup location.

There are two ways to make a reverse address search. One way is to query the [Azure Maps Reverse Address Search API](#) through a service module. The other way is to use the [Fetch API](#) to make a request to the [Azure Maps Reverse Address Search API](#) to find an address. Both ways are surveyed below.

Make a reverse search request via service module

<https://codepen.io/azuremaps/embed/ejEYMZ/?height=265&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

In the code above, the first block constructs a map object and sets the authentication mechanism to use the access token. You can see [create a map](#) for instructions.

The second code block creates a `TokenCredential` to authenticate HTTP requests to Azure Maps with the access token. It then passes the `TokenCredential` to `atlas.service.MapsURL.newPipeline()` and creates a `Pipeline` instance. The `searchURL` represents a URL to Azure Maps [Search](#) operations.

The third code block updates the style of mouse cursor to a pointer and creates a `popup` object. You can see [add a popup on the map](#) for instructions.

The fourth block of code adds a mouse click [event listener](#). When triggered, it creates a search query with the coordinates of the clicked point. It then uses the `getSearchAddressReverse` method to query the [Get Search Address Reverse API](#) for the address of the coordinates. A GeoJSON feature collection is then extracted using the `geojson.getFeatures()` method from the response.

The fifth block of code sets up the HTML popup content to display the response address for the clicked coordinate position.

The change of cursor, the popup object, and the click event are all created in the map's [load event listener](#). This code structure ensures map fully loads before retrieving the coordinates information.

Make a reverse search request via Fetch API

Click on the map to make a reverse geocode request for that location using fetch.

<https://codepen.io/azuremaps/embed/ddXzoB/?height=516&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

In the code above, the first block of code constructs a map object and sets the authentication mechanism to use the access token. You can see [create a map](#) for instructions.

The second block of code updates the style of the mouse cursor to a pointer. It instantiates a `popup` object. You can see [add a popup on the map](#) for instructions.

The third block of code adds an event listener for mouse clicks. Upon a mouse click, it uses the [Fetch API](#) to query the [Azure Maps Reverse Address Search API](#) for the clicked coordinates address. For a successful response, it collects the address for the clicked location. It defines the popup content and position using the `setOptions` function of the `popup` class.

The change of cursor, the popup object, and the click event are all created in the map's [load event listener](#). This code structure ensures the map fully loads before retrieving the coordinates information.

Next steps

[Best practices for using the search service](#)

Learn more about the classes and methods used in this article:

[Map](#)

[Popup](#)

See the following articles for full code examples:

[Show directions from A to B](#)

[Show traffic](#)

Show directions from A to B

11/2/2020 • 3 minutes to read • [Edit Online](#)

This article shows you how to make a route request and show the route on the map.

There are two ways to do so. The first way is to query the [Azure Maps Route API](#) through a service module. The second way is to use the [Fetch API](#) to make a search request to the [Azure Maps Route API](#). Both ways are discussed below.

Query the route via service module

<https://codepen.io/azuremaps/embed/RBZbep/?height=265&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

In the above code, the first block constructs a map object and sets the authentication mechanism to use the access token. You can see [create a map](#) for instructions.

The second block of code creates a `TokenCredential` to authenticate HTTP requests to Azure Maps with the access token. It then passes the `TokenCredential` to `atlas.service.MapsURL.newPipeline()` and creates a `Pipeline` instance. The `routeURL` represents a URL to Azure Maps [Route](#) operations.

The third block of code creates and adds a `DataSource` object to the map.

The fourth block of code creates start and end `points` objects and adds them to the `dataSource` object.

A line is a [Feature](#) for [LineString](#). A [LineLayer](#) renders line objects wrapped in the [DataSource](#) as lines on the map. The fourth block of code creates and adds a line layer to the map. See properties of a line layer at [LinestringLayerOptions](#).

A [symbol layer](#) uses texts or icons to render point-based data wrapped in the [DataSource](#). The texts or the icons render as symbols on the map. The fifth block of code creates and adds a symbol layer to the map.

The sixth block of code queries the Azure Maps routing service, which is part of the [service module](#). The [calculateRouteDirections](#) method of the `RouteURL` is used to get a route between the start and end points. A GeoJSON feature collection from the response is then extracted using the `geojson.getFeatures()` method and is added to the datasource. It then renders the response as a route on the map. For more information about adding a line to the map, see [add a line on the map](#).

The last block of code sets the bounds of the map using the Map's [setCamera](#) property.

The route query, data source, symbol, line layers, and camera bounds are created inside the [event listener](#). This code structure ensures the results are displayed only after the map fully loads.

Query the route via Fetch API

<https://codepen.io/azuremaps/embed/zRyNmP/?height=469&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

In the code above, the first block of code constructs a map object and sets the authentication mechanism to use the access token. You can see [create a map](#) for instructions.

The second block of code creates and adds a `DataSource` object to the map.

The third code block creates the start and destination points for the route. Then, it adds them to the data source. You can see [add a pin on the map](#) for instructions about using `addPins`.

A [LineLayer](#) renders line objects wrapped in the [DataSource](#) as lines on the map. The fourth block of code creates and adds a line layer to the map. See properties of a line layer at [LineLayerOptions](#).

A [symbol layer](#) uses text or icons to render point-based data wrapped in the [DataSource](#) as symbols on the map. The fifth block of code creates and adds a symbol layer to the map. See properties of a symbol layer at [SymbolLayerOptions](#).

The next code block creates `SouthWest` and `NorthEast` points from the start and destination points and sets the bounds of the map using the Map's `setCamera` property.

The last block of code uses the [Fetch API](#) to make a search request to the [Azure Maps Route API](#). The response is then parsed. If the response was successful, the latitude and longitude information is used to create an array a line by connecting those points. The line data is then added to data source to render the route on the map. You can see [add a line on the map](#) for instructions.

The route query, data source, symbol, line layers, and camera bounds are created inside the [event listener](#). Again, we want to ensure that results are displayed after the map loads fully.

Next steps

[Best practices for using the routing service](#)

Learn more about the classes and methods used in this article:

[Map](#)

See the following articles for full code examples:

[Show traffic on the map](#)

[Interacting with the map - mouse events](#)

Use the Azure Maps Indoor Maps module

6/1/2021 • 6 minutes to read • [Edit Online](#)

The Azure Maps Web SDK includes the *Azure Maps Indoor* module. The *Azure Maps Indoor* module allows you to render indoor maps created in Azure Maps Creator services.

Prerequisites

1. [Make an Azure Maps account](#)
2. [Create a Creator resource](#)
3. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.
4. Get a `tilesetId` and a `statesetId` by completing the [tutorial for creating Indoor maps](#). You'll need to use these identifiers to render indoor maps with the Azure Maps Indoor Maps module.

Embed the Indoor Maps module

You can install and embed the *Azure Maps Indoor* module in one of two ways.

To use the globally hosted Azure Content Delivery Network version of the *Azure Maps Indoor* module, reference the following JavaScript and Style Sheet references in the `<head>` element of the HTML file:

```
<link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/indoor/0.1/atlas-indoor.min.css"
type="text/css"/>
<script src="https://atlas.microsoft.com/sdk/javascript/indoor/0.1/atlas-indoor.min.js"></script>
```

Or, you can download the *Azure Maps Indoor* module. The *Azure Maps Indoor* module contains a client library for accessing Azure Maps services. Follow the steps below to install and load the *Indoor* module into your web application.

1. Install the [azure-maps-indoor package](#).

```
>npm install azure-maps-indoor
```

2. Reference the *Azure Maps Indoor* module JavaScript and Style Sheet in the `<head>` element of the HTML file:

```
<link rel="stylesheet" href="node_modules/azure-maps-drawing-tools/dist/atlas-indoor.min.css"
type="text/css" />
<script src="node_modules/azure-maps-drawing-tools/dist/atlas-indoor.min.js"></script>
```

Instantiate the Map object

First, create a *Map object*. The *Map object* will be used in the next step to instantiate the *Indoor Manager* object. The code below shows you how to instantiate the *Map object*.

```

const subscriptionKey = "<Your Azure Maps Primary Subscription Key>";

const map = new atlas.Map("map-id", {
    //use your facility's location
    center: [-122.13203, 47.63645],
    //or, you can use bounds: [# west, # south, # east, # north] and replace # with your map's bounds
    style: "blank",
    view: 'Auto',
    authOptions: {
        authType: 'subscriptionKey',
        subscriptionKey: subscriptionKey
    },
    zoom: 19,
});

```

Instantiate the Indoor Manager

To load the indoor tilesets and map style of the tiles, you must instantiate the *Indoor Manager*. Instantiate the *Indoor Manager* by providing the *Map object* and the corresponding `tilesetId`. If you wish to support [dynamic map styling](#), you must pass the `statesetId`. The `statesetId` variable name is case-sensitive. Your code should like the JavaScript below.

```

const tilesetId = "<tilesetId>";
const statesetId = "<statesetId>";

const indoorManager = new atlas.indoor.IndoorManager(map, {
    tilesetId: tilesetId,
    statesetId: statesetId // Optional
});

```

To enable polling of state data you provide, you must provide the `statesetId` and call `indoorManager.setDynamicStyling(true)`. Polling state data lets you dynamically update the state of dynamic properties or *states*. For example, a feature such as room can have a dynamic property (*state*) called `occupancy`. Your application may wish to poll for any *state* changes to reflect the change inside the visual map. The code below shows you how to enable state polling:

```

const tilesetId = "<tilesetId>";
const statesetId = "<statesetId>";

const indoorManager = new atlas.indoor.IndoorManager(map, {
    tilesetId: tilesetId,
    statesetId: statesetId // Optional
});

if (statesetId.length > 0) {
    indoorManager.setDynamicStyling(true);
}

```

Indoor Level Picker Control

The *Indoor Level Picker* control allows you to change the level of the rendered map. You can optionally initialize the *Indoor Level Picker* control via the *Indoor Manager*. Here's the code to initialize the level control picker:

```

const levelControl = new atlas.control.LevelControl({ position: "top-right" });
indoorManager.setOptions({ levelControl });

```

Indoor Events

The *Azure Maps Indoor* module supports *Map object* events. The *Map object* event listeners are invoked when a level or facility has changed. If you want to run code when a level or a facility have changed, place your code inside the event listener. The code below shows how event listeners can be added to the *Map object*.

```
map.events.add("levelchanged", indoorManager, (eventData) => {  
    //code that you want to run after a level has been changed  
    console.log("The level has changed: ", eventData);  
});  
  
map.events.add("facilitychanged", indoorManager, (eventData) => {  
    //code that you want to run after a facility has been changed  
    console.log("The facility has changed: ", eventData);  
});
```

The `eventData` variable holds information about the level or facility that invoked the `levelchanged` or `facilitychanged` event, respectively. When a level changes, the `eventData` object will contain the `facilityId`, the new `levelNumber`, and other metadata. When a facility changes, the `eventData` object will contain the new `facilityId`, the new `levelNumber`, and other metadata.

Example: Use the Indoor Maps Module

This example shows you how to use the *Azure Maps Indoor* module in your web application. Although the example is limited in scope, it covers the basics of what you need to get started using the *Azure Maps Indoor* module. The complete HTML code is below these steps.

1. Use the Azure Content Delivery Network [option](#) to install the *Azure Maps Indoor* module.
2. Create a new HTML file
3. In the HTML header, reference the *Azure Maps Indoor* module JavaScript and style sheet styles.
4. Initialize a *Map object*. The *Map object* supports the following options:
 - `Subscription key` is your Azure Maps primary subscription key.
 - `center` defines a latitude and longitude for your indoor map center location. Provide a value for `center` if you don't want to provide a value for `bounds`. Format should appear as `center : [-122.13315, 47.63637]`.
 - `bounds` is the smallest rectangular shape that encloses the tileset map data. Set a value for `bounds` if you don't want to set a value for `center`. You can find your map bounds by calling the [Tiles List API](#). The Tiles List API returns the `bbox`, which you can parse and assign to `bounds`. Format should appear as `bounds : [# west, # south, # east, # north]`.
 - `style` allows you to set the color of the background. To display a white background, define `style` as "blank".
 - `zoom` allows you to specify the min and max zoom levels for your map.
5. Next, create the *Indoor Manager* module. Assign the *Azure Maps Indoor* `tilesetId`, and optionally add the `statesetId`.
6. Instantiate the *Indoor Level Picker* control.
7. Add *Map object* event listeners.

Your file should now look similar to the HTML below.

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, user-scalable=no" />
    <title>Indoor Maps App</title>

    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css"
type="text/css" />
    <link rel="stylesheet" href="https://atlas.microsoft.com/sdk/javascript/indoor/0.1/atlas-indoor.min.css"
type="text/css"/>

    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.js"></script>
    <script src="https://atlas.microsoft.com/sdk/javascript/indoor/0.1/atlas-indoor.min.js"></script>

    <style>
      html,
      body {
        width: 100%;
        height: 100%;
        padding: 0;
        margin: 0;
      }

      #map-id {
        width: 100%;
        height: 100%;
      }
    </style>
  </head>

  <body>
    <div id="map-id"></div>
    <script>
      const subscriptionKey = "<Your Azure Maps Primary Subscription Key>";
      const tilesetId = "<your tilesetId>";
      const statesetId = "<your statesetId>";

      const map = new atlas.Map("map-id", {
        //use your facility's location
        center: [-122.13315, 47.63637],
        //or, you can use bounds: [# west, # south, # east, # north] and replace # with your Map bounds
        style: "blank",
        view: 'Auto',
        authOptions: {
          authType: 'subscriptionKey',
          subscriptionKey: subscriptionKey
        },
        zoom: 19,
      });

      const levelControl = new atlas.control.LevelControl({
        position: "top-right",
      });

      const indoorManager = new atlas.indoor.IndoorManager(map, {
        levelControl: levelControl, //level picker
        tilesetId: tilesetId,
        statesetId: statesetId // Optional
      });

      if (statesetId.length > 0) {
        indoorManager.setDynamicStyling(true);
      }

      map.events.add("levelchanged", indoorManager, (eventData) => {
        //put code that runs after a level has been changed
        console.log("The level has changed:", eventData);
      })
    </script>
  </body>
</html>

```

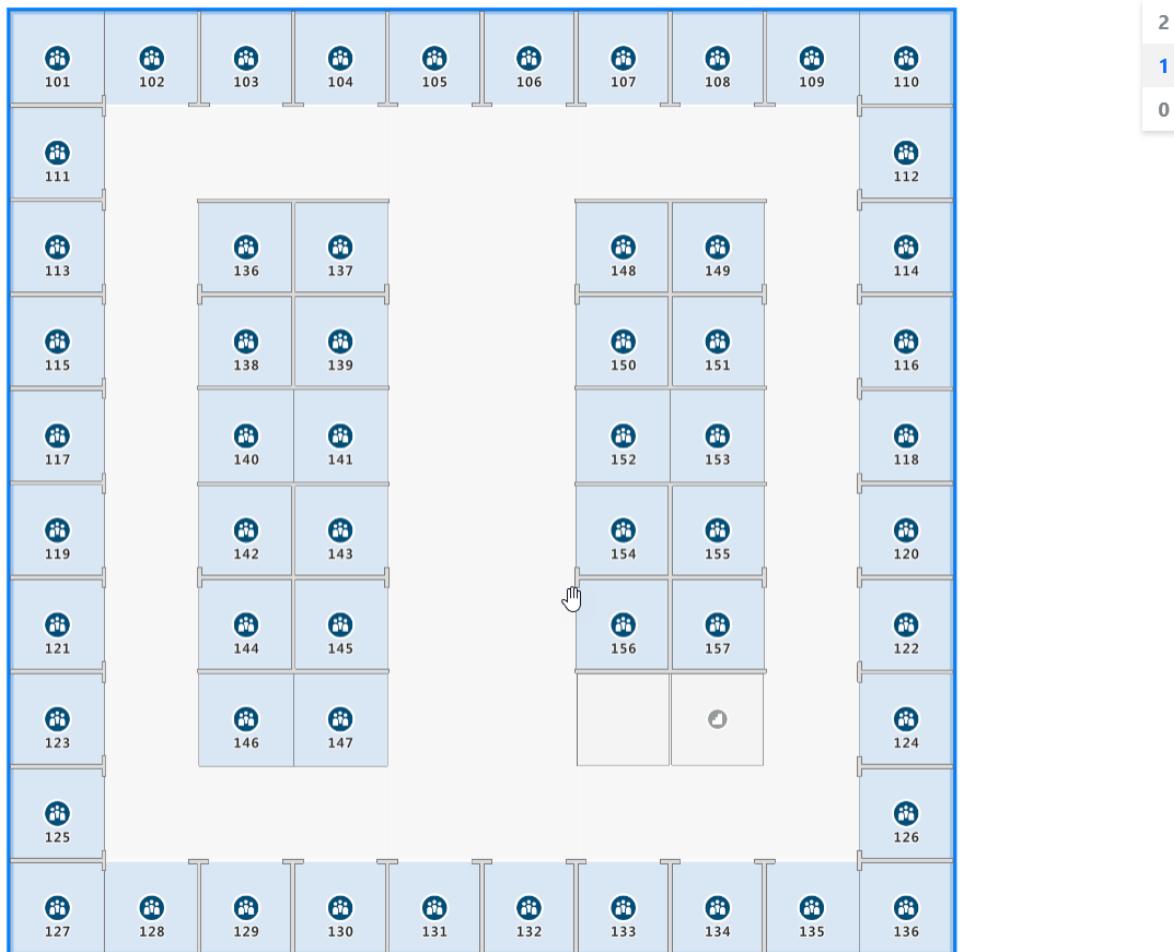
```

});
```

```

map.events.add("facilitychanged", indoorManager, (eventData) => {
    //put code that runs after a facility has been changed
    console.log("The facility has changed:", eventData);
});
</script>
</body>
</html>
```

To see your indoor map, load it into a web browser. It should appear like the image below. If you click on the stairwell feature, the *level picker* will appear in the upper right-hand corner.



[See live demo](#)

Next steps

Read about the APIs that are related to the *Azure Maps Indoor* module:

[Drawing package requirements](#)

[Creator for indoor maps](#)

Learn more about how to add more data to your map:

[Indoor Maps dynamic styling](#)

[Code samples](#)

Implement dynamic styling for Creator indoor maps

6/1/2021 • 3 minutes to read • [Edit Online](#)

You can use Azure Maps Creator [Feature State service](#) to apply styles that are based on the dynamic properties of indoor map data features. For example, you can render facility meeting rooms with a specific color to reflect occupancy status. This article describes how to dynamically render indoor map features with the [Feature State service](#) and the [Indoor Web module](#).

Prerequisites

1. [Create an Azure Maps account](#)
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.
3. [Create a Creator resource](#)
4. Download the [sample Drawing package](#).
5. [Create an indoor map](#) to obtain a `tilesetId` and `statesetId`.
6. Build a web application by following the steps in [How to use the Indoor Map module](#).

This tutorial uses the [Postman](#) application, but you may choose a different API development environment.

Implement dynamic styling

After you complete the prerequisites, you should have a simple web application configured with your subscription key, `tilesetId`, and `statesetId`.

Select features

To implement dynamic styling, a feature - such as a meeting or conference room - must be referenced by its feature `id`. You use the feature `id` to update the dynamic property or *state* of that feature. To view the features defined in a dataset, you can use one of the following methods:

- WFS API (Web Feature service). You can use the [WFS API](#) to query datasets. WFS follows the [Open Geospatial Consortium API Features](#). The WFS API is helpful for querying features within a dataset. For example, you can use WFS to find all mid-size meeting rooms of a specific facility and floor level.
- Implement customized code that a user can use to select features on a map using your web application. We use this option in this article.

The following script implements the mouse-click event. The code retrieves the feature `id` based on the clicked point. In your application, you can insert the code after your Indoor Manager code block. Run your application, and then check the console to obtain the feature `id` of the clicked point.

```
/* Upon a mouse click, log the feature properties to the browser's console. */
map.events.add("click", function(e){

    var features = map.layers.getRenderedShapes(e.position, "indoor");

    features.forEach(function (feature) {
        if (feature.layer.id == 'indoor_unit_office') {
            console.log(feature);
        }
    });
});
```

The [Create an indoor map](#) tutorial configured the feature stateset to accept state updates for `occupancy`.

In the next section, we'll set the occupancy *state* of office `UNIT26` to `true` and office `UNIT27` to `false`.

Set occupancy status

We'll now update the state of the two offices, `UNIT26` and `UNIT27`:

1. In the Postman app, select **New**.
2. In the **Create New** window, select **Collection**.
3. Select **New** again.
4. In the **Create New** window, select **Request**.
5. Enter a **Request name** for the request, such as *POST Data Upload*.
6. Select the collection you previously created, and then select **Save**.
7. Enter the following URL to the [Feature Update States API](#) (replace `{Azure-Maps-Primary-Subscription-key}` with your primary subscription key and `statesetId` with the `statesetId`):

```
https://us.atlas.microsoft.com/featurestatesets/{statesetId}/featureStates/UNIT26?api-version=2.0&subscription-key={Azure-Maps-Primary-Subscription-key}
```

8. Select the **Headers** tab.
9. In the **KEY** field, select `Content-Type`. In the **VALUE** field, select `application/json`.

The screenshot shows the Postman interface with the 'Headers' tab selected. There is one header entry: 'Content-Type' with the value 'application/json'. The 'Content-Type' row is highlighted with a red border.

KEY	VALUE	DESCRIPTION
Content-Type	application/json	
Key	Value	Description

10. Select the **Body** tab.
11. In the dropdown lists, select **raw** and **JSON**.
12. Copy the following JSON style, and then paste it in the **Body** window:

```
{
  "states": [
    {
      "keyName": "occupied",
      "value": true,
      "eventTimestamp": "2020-11-14T17:10:20"
    }
  ]
}
```

IMPORTANT

The update will be saved only if the posted time stamp is after the time stamp used in previous feature state update requests for the same feature `ID`.

13. Change the URL you used in step 7 by replacing `UNIT26` with `UNIT27`:

```
https://us.atlas.microsoft.com/featurestatesets/{statesetId}/featureStates/UNIT27?api-version=2.0&subscription-key={Azure-Maps-Primary-Subscription-key}
```

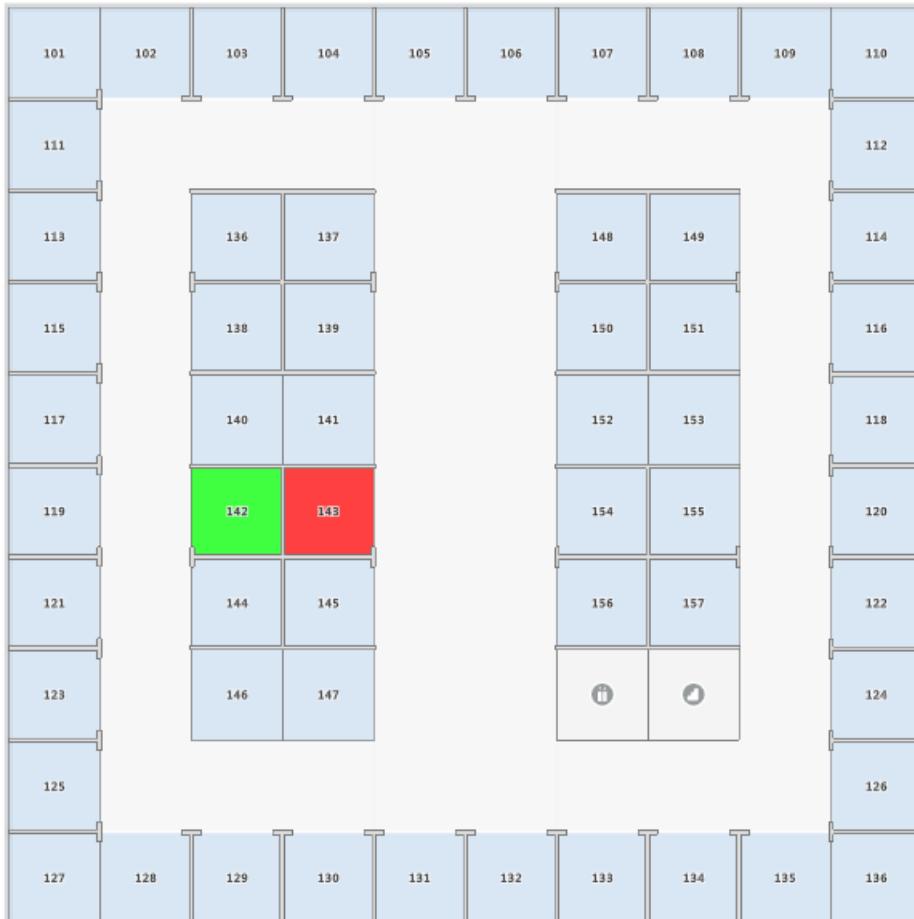
14. Copy the following JSON style, and then paste it in the **Body** window:

```
{  
    "states": [  
        {  
            "keyName": "occupied",  
            "value": false,  
            "eventTimestamp": "2020-11-14T17:10:20"  
        }  
    ]  
}
```

Visualize dynamic styles on a map

The web application that you previously opened in a browser should now reflect the updated state of the map features:

- Office `UNIT27` (142) should appear green.
- Office `UNIT26` (143) should appear red.



[See live demo](#)

Next steps

Learn more by reading:

[Creator for indoor mapping](#)

See the references for the APIs mentioned in this article:

[Data Upload](#)

[Data Conversion](#)

[Dataset](#)

[Tilesset](#)

[Feature State set](#)

[WFS service](#)

Using the Azure Maps Drawing Error Visualizer with Creator

6/1/2021 • 3 minutes to read • [Edit Online](#)

The Drawing Error Visualizer is a stand-alone web application that displays [Drawing package warnings and errors](#) detected during the conversion process. The Error Visualizer web application consists of a static page that you can use without connecting to the internet. You can use the Error Visualizer to fix errors and warnings in accordance with [Drawing package requirements](#). The [Azure Maps Conversion API](#) returns a response with a link to the Error Visualizer only when an error is detected.

Prerequisites

Before you can download the Drawing Error Visualizer, you'll need to:

1. [Create an Azure Maps account](#)
2. [Obtain a primary subscription key](#), also known as the primary key or the subscription key.
3. [Create a Creator resource](#)

This tutorial uses the [Postman](#) application, but you may choose a different API development environment.

Download

1. Upload your Drawing package to the Azure Maps Creator service to obtain a `udid` for the uploaded package. For steps on how to upload a package, see [Upload a drawing package](#).
2. Now that the Drawing package is uploaded, we'll use `udid` for the uploaded package to convert the package into map data. For steps on how to convert a package, see [Convert a drawing package](#).

NOTE

If your conversion process succeeds, you will not receive a link to the Error Visualizer tool.

3. Under the response **Headers** tab in the Postman application, look for the `diagnosticPackageLocation` property, returned by the Conversion API. The response should appear like the following JSON:

```
{  
    "operationId": "77dc9262-d3b8-4e32-b65d-74d785b53504",  
    "created": "2020-04-22T19:39:54.9518496+00:00",  
    "status": "Failed",  
    "properties": {  
        "diagnosticPackageLocation": "https://us.atlas.microsoft.com/mapData/ce61c3c1-faa8-75b7-349f-d863f6523748?api-version=2.0"  
    }  
}
```

4. Download the Drawing Package Error Visualizer by making a `HTTP-GET` request on the `diagnosticPackageLocation` URL.

Setup

Inside the downloaded zipped package from the [diagnosticPackageLocation](#) link, you'll find two files.

- *VisualizationTool.zip*: Contains the source code, media, and web page for the Drawing Error Visualizer.
- *ConversionWarningsAndErrors.json*: Contains a formatted list of warnings, errors, and other details that are used by the Drawing Error Visualizer.

Unzip the *VisualizationTool.zip* folder. It contains the following items:

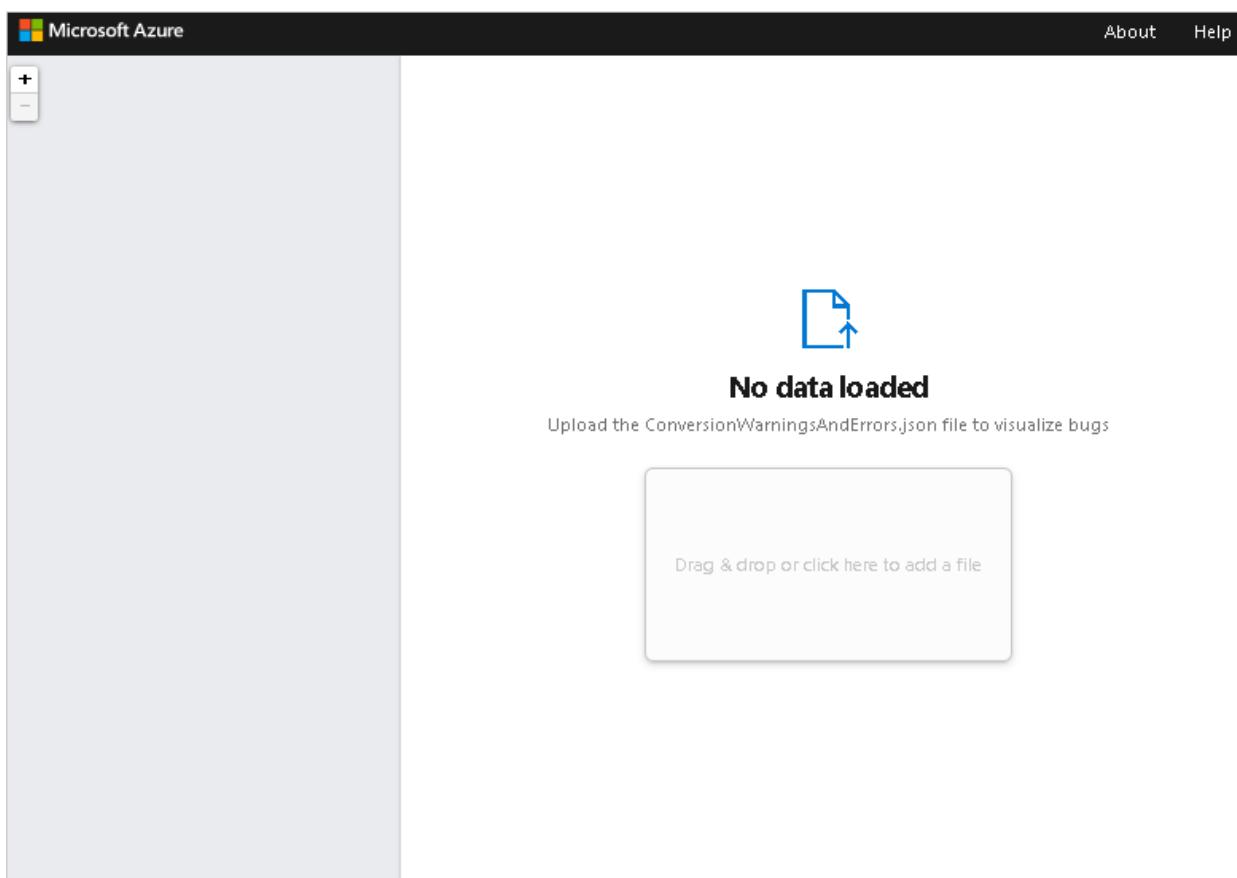
- *assets* folder: contains images and media files
- *static* folder: source code
- *index.html* file: the web application.

Open the *index.html* file using any of the browsers below, with the respective version number. You may use a different version, if the version offers equally compatible behavior as the listed version.

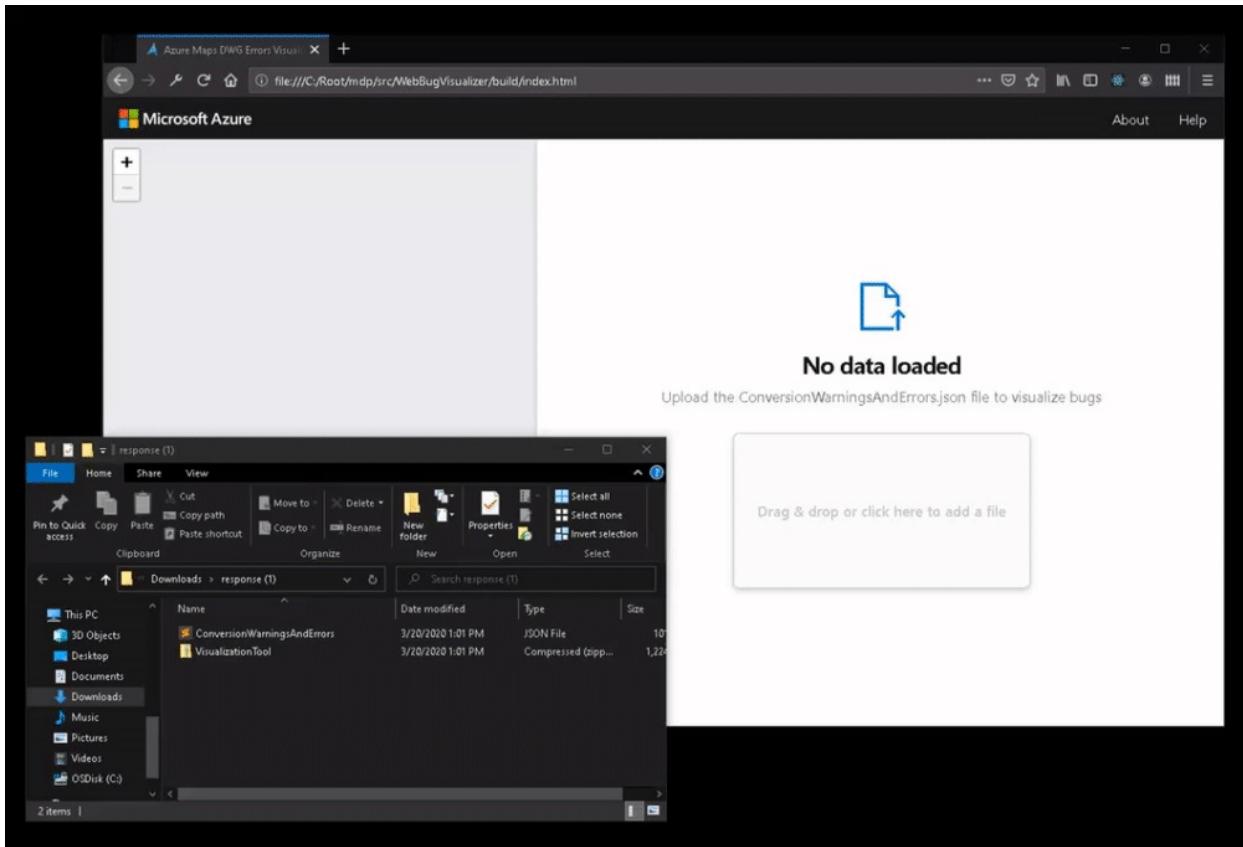
- Microsoft Edge 80
- Safari 13
- Chrome 80
- Firefox 74

Using the Drawing Error Visualizer tool

After launching the Drawing Error Visualizer tool, you'll be presented with the upload page. The upload page contains a drag and drop box. The drag & drop box also functions as button that launches a File Explorer dialog.



The *ConversionWarningsAndErrors.json* file has been placed at the root of the downloaded directory. To load the *ConversionWarningsAndErrors.json*, drag & drop the file onto the box. Or, click on the box, find the file in the [File Explorer dialogue](#), and upload the file.



Once the *ConversionWarningsAndErrors.json* file loads, you'll see a list of your Drawing package errors and warnings. Each error or warning is specified by the layer, level, and a detailed message. To view detailed information about an error or warning, click on the **Details** link. An intractable section will then appear below the list. You may now navigate to each error to learn more details on how to resolve the error.

Code	Message	Level	Layer
automaticRepair...	Automatically repaired invalid polygon. POLYGON ((1...	0	A-DOOR-EXST
unsupportedFeat...	Ignoring unsupported feature representation Spline	0	A-DOOR-EXST
<input checked="" type="checkbox"/> automaticRepair...	Automatically repaired invalid polygon. POLYGON ((1...	0	A-WALL-EXST
automaticRepair...	Automatically repaired invalid polygon. POLYGON ((1...	0	A-DOOR-EXST
geometryWarning	Geometry must have at least 4 points with a tolerance...	0	A-DOOR-CORE-EXST
geometryWarning	Geometry must have at least 4 points with a tolerance...	0	A-DOOR-CORE-EXST
automaticRepair...	Automatically repaired invalid polygon. POLYGON ((1...	0	A-DOOR-CORE-EXST

automaticRepairPerformed ×

```

Automatic repair performed invalid polygon. POLYGON ((14816.443669206277 79216.725753507853, 15610.0343729...
    "root": { 3 items
        "code" : string "automaticRepairPerformed"
        "message" : string "Automatically repaired invalid polygon. POLYGON ((14816.443669206277 79216.725753507853, 15610.0343729...
        "innererror" : { 6 items
            "levelOrdinal" : int 0
            "layerName" : string "A-WALL-EXST"
            "geometry" :
            string "MULTIPOINT ((-122.1336523770226 47.636291469133418), (-122.13364...)"
            "message" : string "Problem with geometry."
            "code" : NULL
            "innererror" : NULL
        }
    }
}

```

Next steps

Once your [Drawing package meets the requirements](#), you can use the [Azure Maps Dataset service](#) to convert the Drawing package to a dataset. Then, you can use the Indoor Maps web module to develop your application. Learn more by reading the following articles:

[Drawing Conversion error codes](#)

[Drawing Package Guide](#)

[Creator for indoor maps](#)

[Use the Indoor Maps module](#)

[Implement indoor map dynamic styling](#)

How to use the Azure Maps Spatial IO module

3/5/2021 • 5 minutes to read • [Edit Online](#)

The Azure Maps Web SDK provides the **Spatial IO module**, which integrates spatial data with the Azure Maps web SDK using JavaScript or TypeScript. The robust features in this module allow developers to:

- [Read and write common spatial data files](#). Supported file formats include: KML, KMZ, GPX, GeoRSS, GML, GeoJSON and CSV files containing columns with spatial information. Also supports Well-Known Text (WKT).
- [Connect to Open Geospatial Consortium \(OGC\) services and integrate with Azure Maps web SDK](#). Overlay Web Map Services (WMS) and Web Map Tile Services (WMTS) as layers on the map.
- [Query data in a Web Feature Service \(WFS\)](#).
- [Overlay complex data sets that contain style information and have them render automatically using minimal code](#).
- [Leverage high-speed XML and delimited file reader and writer classes](#).

In this guide, we'll learn how to integrate and use the Spatial IO module in a web application.

This video provides an overview of Spatial IO module in the Azure Maps Web SDK.

WARNING

Only use data and services that are from a source you trust, especially if referencing it from another domain. The spatial IO module does take steps to minimize risk, however the safest approach is to not allow any dangerous data into your application to begin with.

Prerequisites

Before you can use the Spatial IO module, you'll need to [make an Azure Maps account](#) and [get the primary subscription key for your account](#).

Installing the Spatial IO module

You can load the Azure Maps spatial IO module using one of the two options:

- The globally hosted Azure CDN for the Azure Maps spatial IO module. For this option, you add a reference to the JavaScript in the `<head>` element of the HTML file.

```
<script src="https://atlas.microsoft.com/sdk/javascript/spatial/0/atlas-spatial.js"></script>
```

- The source code for [azure-maps-spatial-io](#) can be loaded locally, and then hosted with your app. This package also includes TypeScript definitions. For this option, use the following command to install the package:

```
npm install azure-maps-spatial-io
```

Then, add a reference to the JavaScript in the `<head>` element of the HTML document:

```
<script src="node_modules/azure-maps-spatial-io/dist/atlas-spatial.min.js"></script>
```

Using the Spatial IO module

1. Create a new HTML file.
2. Load the Azure Maps Web SDK and initialize the map control. See the [Azure Maps map control](#) guide for the details. Once you're done with this step, your HTML file should look like this:

```
<!DOCTYPE html>
<html>

<head>
    <title></title>

    <meta charset="utf-8">

    <!-- Ensures that IE and Edge uses the latest version and doesn't emulate an older version -->
    <meta http-equiv="x-ua-compatible" content="IE=Edge">

    <!-- Ensures the web page looks good on all screen sizes. -->
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->
    <link rel="stylesheet"
        href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css" />
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.js"></script>

    <script type='text/javascript'>

        var map;

        function GetMap() {
            //Initialize a map instance.
            map = new atlas.Map('myMap', {
                view: 'Auto',

                //Add your Azure Maps subscription key to the map SDK. Get an Azure Maps key at
                //https://azure.com/maps
                authOptions: {
                    authType: 'subscriptionKey',
                    subscriptionKey: '<Your Azure Maps Key>'
                }
            });

            //Wait until the map resources are ready.
            map.events.add('ready', function() {

                // Write your code here to make sure it runs once the map resources are ready

            });
        }
    </script>
</head>

<body onload="GetMap()">
    <div id="myMap"></div>
</body>

</html>
```

3. Load the Azure Maps spatial IO module. For this exercise, use the CDN for the Azure Maps spatial IO module. Add the reference below to the `<head>` element of your HTML file:

```
<script src="https://atlas.microsoft.com/sdk/javascript/spatial/0/atlas-spatial.js"></script>
```

4. Initialize a `datasource`, and add the data source to the map. Initialize a `layer`, and add the data source to the map layer. Then, render both the data source and the layer. Before you scroll down to see the full code in the next step, think about the best places to put the data source and layer code snippets. Recall that, before we programmatically manipulate the map, we should wait until the map resource are ready.

```
var datasource, layer;
```

and

```
//Create a data source and add it to the map.  
datasource = new atlas.source.DataSource();  
map.sources.add(datasource);  
  
//Add a simple data layer for rendering the data.  
layer = new atlas.layer.SimpleDataLayer(datasource);  
map.layers.add(layer);
```

5. Putting it all together, your HTML code should look like the following code. This sample demonstrates how to read an XML file from a URL. Then, load and display the file's feature data on the map.

```
<!DOCTYPE html>  
<html>  
  
<head>  
    <title>Spatial IO Module Example</title>  
  
    <meta charset="utf-8">  
  
    <!-- Ensures that IE and Edge uses the latest version and doesn't emulate an older version -->  
    <meta http-equiv="x-ua-compatible" content="IE=Edge">  
  
    <!-- Ensures the web page looks good on all screen sizes. -->  
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  
  
    <!-- Add references to the Azure Maps Map control JavaScript and CSS files. -->  
    <link rel="stylesheet"  
        href="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.min.css" type="text/css" />  
    <script src="https://atlas.microsoft.com/sdk/javascript/mapcontrol/2/atlas.js"></script>  
  
    <!-- Add reference to the Azure Maps Spatial IO module. -->  
    <script src="https://atlas.microsoft.com/sdk/javascript/spatial/0/atlas-spatial.js"></script>  
  
    <script type='text/javascript'>  
        var map, datasource, layer;  
  
        function GetMap() {  
            //Initialize a map instance.  
            map = new atlas.Map('myMap', {  
                view: 'Auto',  
  
                //Add your Azure Maps subscription key to the map SDK. Get an Azure Maps key at  
                //https://azure.com/maps  
                authOptions: {  
                    authType: 'subscriptionKey',  
                    subscriptionKey: '<Your Azure Maps Key>'  
                }  
            });  
        }  
    </script>
```

```

        });

        //Wait until the map resources are ready.
        map.events.add('ready', function() {

            //Create a data source and add it to the map.
            datasource = new atlas.source.DataSource();
            map.sources.add(datasource);

            //Add a simple data layer for rendering the data.
            layer = new atlas.layer.SimpleDataLayer(datasource);
            map.layers.add(layer);

            //Read an XML file from a URL or pass in a raw XML string.
            atlas.io.read('superCoolKmlFile.xml').then(r => {
                if (r) {
                    //Add the feature data to the data source.
                    datasource.add(r);

                    //If bounding box information is known for data, set the map view to it.
                    if (r.bbox) {
                        map.setCamera({
                            bounds: r.bbox,
                            padding: 50
                        });
                    }
                }
            });
        });
    
```

```

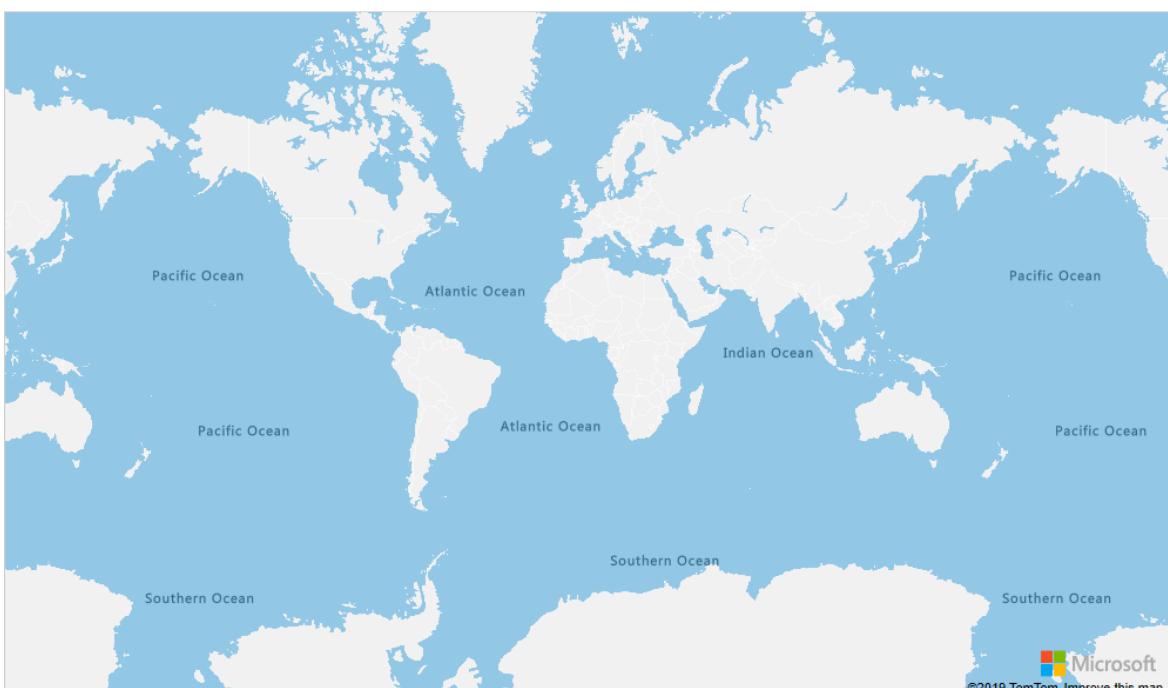
</script>
</head>

<body onload="GetMap()">
    <div id="myMap"></div>
</body>

</html>

```

6. Remember to replace <Your Azure Maps Key> with your primary key. Open your HTML file, and you'll see results similar to the following image:



Next steps

The feature we demonstrated here is only one of the many features available in the Spatial IO module. Read the guides below to learn how to use other functionalities in the Spatial IO module:

[Add a simple data layer](#)

[Read and write spatial data](#)

[Add an OGC map layer](#)

[Connect to a WFS service](#)

[Leverage core operations](#)

[Supported data format details](#)

Refer to the Azure Maps Spatial IO documentation:

[Azure Maps Spatial IO package](#)

Add a simple data layer

11/2/2020 • 6 minutes to read • [Edit Online](#)

The spatial IO module provides a `SimpleDataLayer` class. This class makes it easy to render styled features on the map. It can even render data sets that have style properties and data sets that contain mixed geometry types. The simple data layer achieves this functionality by wrapping multiple rendering layers and using style expressions. The style expressions search for common style properties of the features inside these wrapped layers. The `atlas.io.read` function and the `atlas.io.write` function use these properties to read and write styles into a supported file format. After adding the properties to a supported file format, the file can be used for various purposes. For example, the file can be used to display the styled features on the map.

In addition to styling features, the `SimpleDataLayer` provides a built-in popup feature with a popup template. The popup displays when a feature is clicked. The default popup feature can be disabled, if desired. This layer also supports clustered data. When a cluster is clicked, the map will zoom into the cluster and expand it into individual points and subclusters.

The `SimpleDataLayer` class is intended to be used on large data sets with many geometry types and many styles applied on the features. When used, this class adds an overhead of six layers containing style expressions. So, there are cases when it's more efficient to use the core rendering layers. For example, use a core layer to render a couple of geometry types and a few styles on a feature

Use a simple data layer

The `SimpleDataLayer` class is used like the other rendering layers are used. The code below shows how to use a simple data layer in a map:

```
//Create a data source and add it to the map.  
var datasource = new atlas.source.DataSource();  
map.sources.add(datasource);  
  
//Add a simple data layer for rendering data.  
var layer = new atlas.layer.SimpleDataLayer(datasource);  
map.layers.add(layer);
```

Add features to the data source. Then, the simple data layer will figure out how best to render the features. Styles for individual features can be set as properties on the feature. The following code shows a GeoJSON point feature with a `color` property set to `red`.

```
{  
  "type": "Feature",  
  "geometry": {  
    "type": "Point",  
    "coordinates": [0, 0]  
  },  
  "properties": {  
    "color": "red"  
  }  
}
```

The following code renders the above point feature using the simple data layer.

<https://codepen.io/azuremaps/embed/zYGzpQV/?height=500&theme-id=0&default-tab=js,result&editable=true&rerun-position=hidden&>

The real power of the simple data layer comes when:

- There are several different types of features in a data source; or
- Features in the data set have several style properties individually set on them; or
- You're not sure what the data set exactly contains.

For example when parsing XML data feeds, you may not know the exact styles and geometry types of the features. The following sample shows the power of the simple data layer by rendering the features of a KML file. It also demonstrates various options that the simple data layer class provides.

<https://codepen.io/azuremaps/embed/gOpRXgy/?height=700&theme-id=0&default-tab=result&rerun-position=hidden&>

NOTE

This simple data layer uses the `popup template` class to display KML balloons or feature properties as a table. By default, all content rendered in the popup will be sandboxed inside of an iframe as a security feature. However, there are limitations:

- All scripts, forms, pointer lock and top navigation functionality is disabled. Links are allowed to open up in a new tab when clicked.
- Older browsers that don't support the `srcdoc` parameter on iframes will be limited to rendering a small amount of content.

If you trust the data being loaded into the popups and potentially want these scripts loaded into popups be able to access your application, you can disable this by setting the `popup templates` `sandboxContent` option to false.

Default supported style properties

As mentioned earlier, the simple data layer wraps several of the core rendering layers: bubble, symbol, line, polygon, and extruded polygon. It then uses expressions to search for valid style properties on individual features.

Azure Maps and GitHub style properties are the two main sets of supported property names. Most property names of the different Azure maps layer options are supported as style properties of features in the simple data layer. Expressions have been added to some layer options to support style property names that are commonly used by GitHub. These property names are defined by [GitHub's GeoJSON map support](#), and they're used to style GeoJSON files that are stored and rendered within the platform. All of GitHub's styling properties are supported in the simple data layer, except the `marker-symbol` styling properties.

If the reader comes across a less common style property, it will convert it to the closest Azure Maps style property. Additionally, the default style expressions can be overridden by using the `getLayers` function of the simple data layer and updating the options on any of the layers.

The following sections provide details on the default style properties that are supported by the simple data layer. The order of the supported property name is also the priority of the property. If two style properties are defined for the same layer option, then the first one in the list has higher precedence. Colors can be any CSS3 color value; HEX, RGB, RGBA, HSL, HSLA, or named color value.

Bubble layer style properties

If a feature is a `Point` or a `MultiPoint`, and the feature doesn't have an `image` property that would be used as a custom icon to render the point as a symbol, then the feature will be rendered with a `BubbleLayer`.

LAYER OPTION	SUPPORTED PROPERTY NAME(S)	DEFAULT VALUE
color	color, marker-color	'#1A73AA'
radius	size ¹ , marker-size ² , scale ¹	8
strokeColor	strokeColor, stroke	'#FFFFFF'

[1] The size and scale values are considered scalar values, and they'll be multiplied by 8

[2] If the GitHub marker-size option is specified, then the following values will be used for the radius.

MARKER SIZE	RADIUS
small	6
medium	8
large	12

Clusters are also rendered using the bubble layer. By default the radius of a cluster is set to 16. The color of the cluster varies depending on the number of points in the cluster, as defined below:

# OF POINTS	COLOR
>= 100	red
>= 10	yellow
< 10	green

Symbol style properties

If a feature is a Point or a MultiPoint, and the feature has an image property that would be used as a custom icon to render the point as a symbol, then the feature will be rendered with a symbolLayer.

LAYER OPTION	SUPPORTED PROPERTY NAME(S)	DEFAULT VALUE
image	image	none
size	size, marker-size ¹	1
rotation	rotation	0
offset	offset	[0, 0]
anchor	anchor	'bottom'

[1] If the GitHub marker-size option is specified, then the following values will be used for the icon size option.

MARKER SIZE	SYMBOL SIZE
small	0.5
medium	1
large	2

If the point feature is a cluster, the `point_count_abbreviated` property will be rendered as a text label. No image will be rendered.

Line style properties

If the feature is a `LineString`, `MultiLineString`, `Polygon`, or `MultiPolygon`, then the feature will be rendered with a `LineLayer`.

LAYER OPTION	SUPPORTED PROPERTY NAME(S)	DEFAULT VALUE
<code>strokeColor</code>	<code>strokeColor</code> , <code>stroke</code>	'#1E90FF'
<code>strokeWidth</code>	<code>strokeWidth</code> , <code>stroke-width</code> , <code>stroke-thickness</code>	3
<code>strokeOpacity</code>	<code>strokeOpacity</code> , <code>stroke-opacity</code>	1

Polygon style properties

If the feature is a `Polygon` or a `MultiPolygon`, and the feature either doesn't have a `height` property or the `height` property is zero, then the feature will be rendered with a `PolygonLayer`.

LAYER OPTION	SUPPORTED PROPERTY NAME(S)	DEFAULT VALUE
<code>fillColor</code>	<code>fillColor</code> , <code>fill</code>	'#1E90FF'
<code>fillOpacity</code>	<code>fillOpacity</code> , 'fill-opacity'	0.5

Extruded polygon style properties

If the feature is a `Polygon` or a `MultiPolygon`, and has a `height` property with a value greater than 0, the feature will be rendered with an `PolygonExtrusionLayer`.

LAYER OPTION	SUPPORTED PROPERTY NAME(S)	DEFAULT VALUE
<code>base</code>	<code>base</code>	0
<code>fillColor</code>	<code>fillColor</code> , <code>fill</code>	'#1E90FF'
<code>height</code>	<code>height</code>	0

Next steps

Learn more about the classes and methods used in this article:

[SimpleDataLayer](#)

SimpleDataLayerOptions

See the following articles for more code samples to add to your maps:

[Read and write spatial data](#)

[Add an OGC map layer](#)

[Connect to a WFS service](#)

[Leverage core operations](#)

[Supported data format details](#)

Read and write spatial data

11/2/2020 • 7 minutes to read • [Edit Online](#)

The table below lists the spatial file formats that are supported for reading and writing operations with the Spatial IO module.

DATA FORMAT	READ	WRITE
GeoJSON	✓	✓
GeoRSS	✓	✓
GML	✓	✓
GPX	✓	✓
KML	✓	✓
KMZ	✓	✓
Spatial CSV	✓	✓
Well-Known Text	✓	✓

These next sections outline all the different tools for reading and writing spatial data using the Spatial IO module.

Read spatial data

The `atlas.io.read` function is the main function used to read common spatial data formats such as KML, GPX, GeoRSS, GeoJSON, and CSV files with spatial data. This function can also read compressed versions of these formats, as a zip file or a KMZ file. The KMZ file format is a compressed version of KML that can also include assets such as images. Alternatively, the read function can take in a URL that points to a file in any of these formats. URLs should be hosted on a CORS enabled endpoint, or a proxy service should be provided in the read options. The proxy service is used to load resources on domains that aren't CORS enabled. The read function returns a promise to add the image icons to the map, and processes data asynchronously to minimize impact to the UI thread.

When reading a compressed file, either as a zip or a KMZ, it will be unzipped and scanned for the first valid file. For example, doc.kml, or a file with other valid extension, such as: .kml, .xml, geojson, .json, .csv, .tsv, or .txt. Then, images referenced in KML and GeoRSS files are preloaded to ensure they're accessible. Inaccessible image data may load an alternative fallback image or will be removed from the styles. Images extracted from KMZ files will be converted to data URIs.

The result from the read function is a `SpatialDataSet` object. This object extends the GeoJSON FeatureCollection class. It can easily be passed into a `DataSource` as-is to render its features on a map. The `SpatialDataSet` not only contains feature information, but it may also include KML ground overlays, processing metrics, and other details as outlined in the following table.

PROPERTY NAME	TYPE	DESCRIPTION
<code>bbox</code>	<code>BoundingBox</code>	Bounding box of all the data in the data set.
<code>features</code>	<code>Feature[]</code>	GeoJSON features within the data set.
<code>groundOverlays</code>	<code>(atlas.layer.ImageLayer atlas.layers.OgcMapLayer)[]</code>	An array of KML GroundOverlays.
<code>icons</code>	<code>Record<string, string></code>	A set of icon URLs. Key = icon name, Value = URL.
<code>properties</code>	<code>any</code>	Property information provided at the document level of a spatial data set.
<code>stats</code>	<code>SpatialDataSetStats</code>	Statistics about the content and processing time of a spatial data set.
<code>type</code>	<code>'FeatureCollection'</code>	Read-only GeoJSON type value.

Examples of reading spatial data

The following code shows how to read a spatial data set, and render it on the map using the `SimpleDataLayer` class. The code uses a GPX file pointed to by a URL.

<https://codepen.io/azuremaps/embed/yLNxRZx/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

The next code demo shows how to read and load KML, or KMZ, to the map. KML can contain ground overlays, which will be in the form of an `ImageLayer` or `OgcMapLayer`. These overlays must be added on the map separately from the features. Additionally, if the data set has custom icons, those icons need to be loaded to the maps resources before the features are loaded.

<https://codepen.io/azuremaps/embed/XWbgwxX/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

You may optionally provide a proxy service for accessing cross domain assets that may not have CORS enabled. The read function will try to access files on another domain using CORS first. After the first time it fails to access any resource on another domain using CORS it will only request additional files if a proxy service has been provided. The read function appends the file URL to the end of the proxy URL provided. This snippet of code shows how to pass a proxy service into the read function:

```
//Read a file from a URL or pass in a raw data as a string.
atlas.io.read('https://nonCorsDomain.example.com/mySuperCoolData.xml', {
    //Provide a proxy service
    proxyService: window.location.origin + '/YourCorsEnabledProxyService.ashx?url='
}).then(async r => {
    if (r) {
        // Some code goes here . . .
    }
});
```

The demo below shows how to read a delimited file and render it on the map. In this case, the code uses a CSV

file that has spatial data columns.

<https://codepen.io/azuremaps/embed/ExjXEB/%3Fheight=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Write spatial data

There are two main write functions in the spatial IO module. The `atlas.io.write` function generates a string, while the `atlas.io.writeCompressed` function generates a compressed zip file. The compressed zip file would contain a text-based file with the spatial data in it. Both of these functions return a promise to add the data to the file. And, they both can write any of the following data: `SpatialDataSet`, `DataSource`, `ImageLayer`, `OgcMapLayer`, feature collection, feature, geometry, or an array of any combination of these data types. When writing using either functions, you can specify the wanted file format. If the file format isn't specified, then the data will be written as KML.

The tool below demonstrates the majority of the write options that can be used with the `atlas.io.write` function.

<https://codepen.io/azuremaps/embed/YzXxXPG/%3Fheight=700&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

Example of writing spatial data

The following sample allows you to drag and drop and then load spatial files on the map. You can export GeoJSON data from the map and write it in one of the supported spatial data formats as a string or as a compressed file.

<https://codepen.io/azuremaps/embed/zYGdGoO/%3Fheight=700&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

You may optionally provide a proxy service for accessing cross domain assets that may not have CORS enabled. This snippet of code shows you could incorporate a proxy service:

```
atlas.io.read(data, {
    //Provide a proxy service
    proxyService: window.location.origin + '/YourCorsEnabledProxyService.ashx?url='
}).then(
    //Success
    function(r) {
        //some code goes here ...
    }
);
```

Read and write Well-Known Text (WKT)

Well-Known Text (WKT) is an Open Geospatial Consortium (OGC) standard for representing spatial geometries as text. Many geospatial systems support WKT, such as Azure SQL and Azure PostgreSQL using the PostGIS plugin. Like most OGC standards, coordinates are formatted as "longitude latitude" to align with the "x y" convention. As an example, a point at longitude -110 and latitude 45 can be written as `POINT(-110 45)` using the WKT format.

Well-known text can be read using the `atlas.io.ogc.WKT.read` function, and written using the `atlas.io.ogc.WKT.write` function.

Examples of reading and writing Well-Known Text (WKT)

The following code shows how to read the well-known text string `POINT(-122.34009 47.60995)` and render it on the map using a bubble layer.

<https://codepen.io/azuremaps/embed/XWbabLd/?height=500&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

The following code demonstrates reading and writing well-known text back and forth.

<https://codepen.io/azuremaps/embed/JjdyYav/?height=700&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

Read and write GML

GML is a spatial XML file specification that's often used as an extension to other XML specifications. GeoJSON data can be written as XML with GML tags using the `atlas.io.core.GmlWriter.write` function. The XML that contains GML can be read using the `atlas.io.core.GmlReader.read` function. The read function has two options:

- The `isAxisOrderLonLat` option - The axis order of coordinates "latitude, longitude" or "longitude, latitude" can vary between data sets, and it isn't always well defined. By default the GML reader reads the coordinate data as "latitude, longitude", but setting this option to true will read it as "longitude, latitude".
- The `propertyTypes` option - This option is a key value lookup table where the key is the name of a property in the data set. The value is the object type to cast the value to when parsing. The supported type values are: `string`, `number`, `boolean`, and `date`. If a property isn't in the lookup table or the type isn't defined, the property will be parsed as a string.

The `atlas.io.read` function will default to the `atlas.io.core.GmlReader.read` function when it detects that the input data is XML, but the data isn't one of the other support spatial XML formats.

The `GmlReader` will parse coordinates that has one of the following SRIDs:

- EPSG:4326 (Preferred)
- EPSG:4269, EPSG:4283, EPSG:4258, EPSG:4308, EPSG:4230, EPSG:4272, EPSG:4271, EPSG:4267, EPSG:4608, EPSG:4674 possibly with a small margin of error.
- EPSG:3857, EPSG:102100, EPSG:3785, EPSG:900913, EPSG:102113, EPSG:41001, EPSG:54004

More resources

Learn more about the classes and methods used in this article:

[atlas.io static functions](#)

[SpatialDataSet](#)

[SpatialDataSetStats](#)

[GmlReader](#)

[GmlWriter](#)

[atlas.io.ogc.WKT functions](#)

[Connect to a WFS service](#)

[Leverage core operations](#)

[Supported data format details](#)

Next steps

See the following articles for more code samples to add to your maps:

[Add an OGC map layer](#)

Add a map layer from the Open Geospatial Consortium (OGC)

11/2/2020 • 2 minutes to read • [Edit Online](#)

The `atlas.layer.OgcMapLayer` class can overlay Web Map Services (WMS) imagery and Web Map Tile Services (WMTS) imagery on the map. WMS is a standard protocol developed by OGC for serving georeferenced map images over the internet. Image georeferencing is the processes of associating an image to a geographical location. WMTS is also a standard protocol developed by OGC. It's designed for serving pre-rendered and georeferenced map tiles.

The following sections outline the web map service features that are supported by the `OgcMapLayer` class.

Web Map Service (WMS)

- Supported versions: `1.0.0`, `1.1.0`, `1.1.1`, and `1.3.0`
- The service must support the `EPSG:3857` projection system, or handle reprojections.
- `GetFeatureInfo` requires the service to support `EPSG:4326` or handle reprojections.
- Supported operations:

OPERATION	DESCRIPTION
GetCapabilities	Retrieves metadata about the service with the supported capabilities
GetMap	Retrieves a map image for a specified region
GetFeatureInfo	Retrieves <code>feature_info</code> , which contains underlying data about the feature

Web Map Tile Service (WMTS)

- Supported versions: `1.0.0`
- Tiles must be square, such that `TileWidth == TileHeight`.
- CRS supported: `EPSG:3857` or `GoogleMapsCompatible`
- `TileMatrix` identifier must be an integer value that corresponds to a zoom level on the map. On an azure map, the zoom level is a value between `"0"` and `"22"`. So, `"0"` is supported, but `"00"` isn't supported.
- Supported operations:

OPERATION	DESCRIPTION
GetCapabilities	Retrieves the supported operations and features
GetTile	Retrieves imagery for a particular tile

Overlay an OGC map layer

The `url` can be the base URL for the service or a full URL with the query for getting the capabilities of the service. Depending on the details provided, the WFS client may try several standard URL formats to determine how to initially access the service.

The following code shows how to overlay an OGC map layer on the map.

<https://codepen.io/azuremaps/embed/xxGLZWB/?height=700&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

OGC map layer options

The below sample demonstrates the different OGC map layer options. You may click on the code pen button at the top-right corner to edit the code pen.

<https://codepen.io/azuremaps/embed/abOyEVQ/?height=700&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

OGC Web Map Service explorer

The following tool overlays imagery from the Web Map Services (WMS) and Web Map Tile Services (WMTS) as layers. You may select which layers in the service are rendered on the map. You may also view the associated legends for these layers.

<https://codepen.io/azuremaps/embed/YzXxYdX/?height=750&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

You may also specify the map settings to use a proxy service. The proxy service lets you load resources that are hosted on domains that don't have CORS enabled.

Next steps

Learn more about the classes and methods used in this article:

[OgcMapLayer](#)

[OgcMapLayerOptions](#)

See the following articles, which contain code samples you could add to your maps:

[Connect to a WFS service](#)

[Leverage core operations](#)

[Supported data format details](#)

Connect to a WFS service

11/2/2020 • 2 minutes to read • [Edit Online](#)

A Web Feature Service (WFS) is a web service for querying spatial data that has a standardized API that is defined by the Open Geospatial Consortium (OGC). The `WfsClient` class in the spatial IO module lets developers connect to a WFS service and query data from the service.

The following features are supported by the `WfsClient` class:

- Supported versions: `1.0.0`, `1.1.0`, and `2.0.0`
- Supported filter operators: binary comparisons, logic, math, value, and `bbox`.
- Requests are made using `HTTP GET` only.
- Supported operations:

OPERATION	DESCRIPTION
GetCapabilities	Generates a metadata document with valid WFS operations and parameters
GetFeature	Returns a selection of features from a data source
DescribeFeatureType	Returns the supported feature types

Using the WFS client

The `atlas.io.ogc.WfsClient` class in the spatial IO module makes it easy to query a WFS service and convert the responses into GeoJSON objects. This GeoJSON object can then be used for other mapping purposes.

The following code queries a WFS service and renders the returned features on the map.

<https://codepen.io/azuremaps/embed/MWwvVYY/?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

Supported filters

The specification for the WFS standard makes use of OGC filters. The filters below are supported by the WFS client, assuming that the service being called also supports these filters. Custom filter strings can be passed into the `CustomFilter` class.

Logical operators

- `And`
- `Or`
- `Not`

Value operators

- `GmlObjectId`
- `ResourceId`

Math operators

- `Add`
- `Sub`
- `Mul`
- `Div`

Comparison operators

- `PropertyIsEqualTo`
- `PropertyIsNotEqualTo`
- `PropertyIsLessThan`
- `PropertyIsGreaterThan`
- `PropertyIsLessThanOrEqual`
- `PropertyIsGreaterThanOrEqual`
- `PropertyIsLike`
- `PropertyIsNull`
- `PropertyIsNil`
- `PropertyIsBetween`

The following code demonstrates the use of different filters with the WFS client.

<https://codepen.io/azuremaps/embed/NWqvYrV/?height=500&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

WFS service explorer

The following code uses the WFS client to explore WFS services. Select a property type layer within the service and see the associated legend.

<https://codepen.io/azuremaps/embed/bGdrvmG/?height=700&theme-id=0&default-tab=result&embed-version=2&editable=true&rerun-position=hidden&>

To access WFS services hosted on non-CORS enabled endpoints, a CORS enabled proxy service can be passed into the `proxyService` option of the WFS client as shown below.

```
//Create the WFS client to access the service and use the proxy service settings
client = new atlas.io.ogc.WfsClient({
    url: url,
    proxyService: window.location.origin + '/YourCorsEnabledProxyService.ashx?url='
});
```

Next steps

Learn more about the classes and methods used in this article:

[WfsClient](#)

[WfsServiceOptions](#)

See the following articles for more code samples to add to your maps:

[Leverage core operations](#)

Supported data format details

Core IO operations

11/2/2020 • 4 minutes to read • [Edit Online](#)

In addition to providing tools to read spatial data files, the spatial IO module exposes core underlying libraries to read and write XML and delimited data fast and efficiently.

The `atlas.io.core` namespace contains two low-level classes that can quickly read and write CSV and XML data. These base classes power the spatial data readers and writers in the Spatial IO module. Feel free to use them to add additional reading and writing support for CSV or XML files.

Read delimited files

The `atlas.io.core.CsvReader` class reads strings that contain delimited data sets. This class provides two methods for reading data:

- The `read` function will read the full data set and return a two-dimensional array of strings representing all cells of the delimited data set.
- The `getNextRow` function reads each line of text in a delimited data set and returns an array of string representing all cells in that line of data set. The user can process the row and dispose any unneeded memory from that row before processing the next row. So, function is more memory efficient.

By default, the reader will use the comma character as the delimiter. However, the delimiter can be changed to any single character or set to `'auto'`. When set to `'auto'`, the reader will analyze the first line of text in the string. Then, it will select the most common character from the table below to use as the delimiter.

DELIMITER	CHARACTER
Comma	,
Tab	\t
Pipe	

This reader also supports text qualifiers that are used to handle cells that contain the delimiter character. The quote (`''''`) character is the default text qualifier, but it can be changed to any single character.

Write delimited files

The `atlas.io.core.CsvWriter` writes an array of objects as a delimited string. Any single character can be used as a delimiter or a text qualifier. The default delimiter is comma (`','`) and the default text qualifier is the quote (`''''`) character.

To use this class, follow the steps below:

- Create an instance of the class and optionally set a custom delimiter or text qualifier.
- Write data to the class using the `write` function or the `writeRow` function. For the `write` function, pass a two-dimensional array of objects representing multiple rows and cells. To use the `writeRow` function, pass an array of objects representing a row of data with multiple columns.
- Call the `toString` function to retrieve the delimited string.
- Optionally, call the `clear` method to make the writer reusable and reduce its resource allocation, or call the

`delete` method to dispose of the writer instance.

NOTE

The number of columns written will be constrained to the number of cells in the first row of the data passed to the writer.

Read XML files

The `atlas.io.core.SimpleXmlReader` class is faster at parsing XML files than `DOMParser`. However, the `atlas.io.core.SimpleXmlReader` class requires XML files to be well formatted. XML files that aren't well formatted, for example missing closing tags, will likely result in an error.

The following code demonstrates how to use the `SimpleXmlReader` class to parse an XML string into a JSON object and serialize it into a desired format.

```
//Create an instance of the SimpleXmlReader and parse an XML string into a JSON object.  
var xmlDoc = new atlas.io.core.SimpleXmlReader().parse(xmlStringToParse);  
  
//Verify that the root XML tag name of the document is the file type your code is designed to parse.  
if (xmlDoc && xmlDoc.root && xmlDoc.root.tagName && xmlDoc.root.tagName === '<Your desired root XML tag  
name>') {  
  
    var node = xmlDoc.root;  
  
    //Loop through the child node tree to navigate through the parsed XML object.  
    for (var i = 0, len = node.childNodes.length; i < len; i++) {  
        childNode = node.childNodes[i];  
  
        switch (childNode.tagName) {  
            //Look for tag names, parse and serialized as desired.  
        }  
    }  
}
```

Write XML files

The `atlas.io.core.SimpleXmlWriter` class writes well-formatted XML in a memory efficient way.

The following code demonstrates how to use the `SimpleXmlWriter` class to generate a well-formatted XML string.

```

//Create an instance of the SimpleXmlWriter class.
var writer = new atlas.io.core.SimpleXmlWriter();

//Start writing the document. All write functions return a reference to the writer, making it easy to chain
//the function calls to reduce the code size.
writer.writeStartDocument(true)
    //Specify the root XML tag name, in this case 'root'
    .writeStartElement('root', {
        //Attributes to add to the root XML tag.
        'version': '1.0',
        'xmlns': 'http://www.example.com',
        //Example of a namespace.
        'xmlns:abc': 'http://www.example.com/abc'
    });
}

//Start writing an element that has the namespace abc and add other XML elements as children.
writer.writeStartElement('abc:parent');

//Write a simple XML element like <title>Azure Maps is awesome!</title>
writer.writeElement('title', 'Azure Maps is awesome!');

//Close the element that we have been writing children to.
writer.writeEndElement();

//Finish writing the document by closing the root tag and the document.
writer.writeEndElement().writeEndDocument();

//Get the generated XML string from the writer.
var xmlString = writer.toString();

```

The generated XML from the above code would look like the following.

```

<?xml version="1.0" encoding="UTF-8"?>
<root version="1.0" xmlns="http://www.example.com" xmlns:abc="http://www.example.com/abc">
    <abc:parent>
        <title>Azure Maps is awesome!</title>
    </abc:parent>
</root>

```

Next steps

Learn more about the classes and methods used in this article:

[CsvReader](#)

[CsvWriter](#)

[SimpleXmlReader](#)

[SimpleXmlWriter](#)

See the following articles for more code samples to add to your maps:

[Supported data format details](#)

Supported data format details

11/2/2020 • 10 minutes to read • [Edit Online](#)

This article provides specifics on the read and write support for all XML tags and Well-Known Text geometry types. It also details how the delimited spatial data is parsed in the spatial IO module.

Supported XML namespaces

The spatial IO module supports XML tags from the following namespaces.

NAMESPACE PREFIX	NAMESPACE URI	NOTES
atom	http://www.w3.org/2005/Atom	
geo	http://www.w3.org/2003/01/geo/wgs84_pos#	Read only support in GeoRSS files.
georss	http://www.georss.org/georss	
geourl	http://geourl.org/rss/module/	Read only support in GeoRSS files.
gml	http://www.opengis.net/gml	
gpx	http://www.topografix.com/GPX/1/1	
gpxx	http://www.garmin.com/xmlschemas/GpxExtensions/v3	Read only support in GPX files. Parses and uses DisplayColor. All other properties added to shape metadata.
gpx_style	http://www.topografix.com/GPX/gpx_style	Supported in GPX files. Uses line color.
gx	http://www.google.com/kml/ext/2.2	
kml	http://www.opengis.net/kml/2.2	
rss		Read only. GeoRSS writes using Atom format.

Supported XML elements

The spatial IO module supports the following XML elements. Any XML tags that aren't supported will be converted into a JSON object. Then, each tag will be added as a property in the `properties` field of the parent shape or layer.

KML elements

The spatial IO module supports the following KML elements.

ELEMENT NAME	READ	WRITE	NOTES
--------------	------	-------	-------

ELEMENT NAME	READ	WRITE	NOTES
<code>address</code>	partial	yes	Object is parsed but isn't used for positioning shape.
<code>AddressDetails</code>	partial	no	Object is parsed but isn't used for positioning shape.
<code>atom:author</code>	yes	yes	
<code>atom:link</code>	yes	yes	
<code>atom:name</code>	yes	yes	
<code>BalloonStyle</code>	partial	partial	<code>displayMode</code> isn't supported. Converted to a <code>PopupTemplate</code> . To write, add a <code>popupTemplate</code> property as a property of the feature you want to write it for.
<code>begin</code>	yes	yes	
<code>color</code>	yes	yes	Includes <code>#AABBGGRR</code> and <code>#BBGGRR</code> . Parsed into a CSS color string
<code>colorMode</code>	yes	no	
<code>coordinates</code>	yes	yes	
<code>Data</code>	yes	yes	
<code>description</code>	yes	yes	
<code>displayName</code>	yes	yes	
<code>Document</code>	yes	yes	
<code>drawOrder</code>	partial	no	Read for ground overlays and used to sort them.
<code>east</code>	yes	yes	
<code>end</code>	yes	yes	
<code>ExtendedData</code>	yes	yes	Supports untyped <code>Data</code> , <code>SimpleData</code> or <code>Schema</code> , and entity replacements of the form <code>#[dataName]</code> .

ELEMENT NAME	READ	WRITE	NOTES
<code>extrude</code>	partial	partial	Only supported for polygons. MultiGeometry that have polygons of different heights will be broken out into individual features. Line styles aren't supported. Polygons with an altitude of 0 will be rendered as a flat polygon. When reading, the altitude of the first coordinate in the exterior ring will be added as a height property of the polygon. Then, the altitude of the first coordinate will be used to render the polygon on the map.
<code>fill</code>	yes	yes	
<code>Folder</code>	yes	yes	
<code>GroundOverlay</code>	yes	yes	<code>color</code> isn't supported
<code>heading</code>	partial	no	Parsed but not rendered by <code>SimpleDataLayer</code> . Only writes if data is stored in the property of the shape.
<code>hotSpot</code>	yes	partial	Only writes if data is stored in the property of the shape. Units are outputted as "pixels" only.
<code>href</code>	yes	yes	
<code>Icon</code>	partial	partial	Parsed but not rendered by <code>SimpleDataLayer</code> . Only writes the icon property of the shape if it contains a URL data. Only <code>href</code> is supported.
<code>IconStyle</code>	partial	partial	<code>icon</code> , <code>heading</code> , <code>colorMode</code> , and <code>hotspots</code> values are parsed, but they aren't rendered by <code>SimpleDataLayer</code>
<code>innerBoundaryIs</code>	yes	yes	
<code>kml</code>	yes	yes	
<code>LabelStyle</code>	no	no	

ELEMENT NAME	READ	WRITE	NOTES
<code>LatLonBox</code>	yes	yes	
<code>gx:LatLonQuad</code>	yes	yes	
<code>LinearRing</code>	yes	yes	
<code>LineString</code>	yes	yes	
<code>LineStyle</code>	yes	yes	<code>colorMode</code> isn't supported.
<code>Link</code>	yes	no	Only the <code>href</code> property is supported for network links.
<code>MultiGeometry</code>	partial	partial	May be broken out into individual features when read.
<code>name</code>	yes	yes	
<code>NetworkLink</code>	yes	no	Links need to be on the same domain as the document.
<code>NetworkLinkControl</code>	no	no	
<code>north</code>	yes	yes	
<code>open</code>	yes	yes	
<code>outerBoundaryIs</code>	yes	yes	
<code>outline</code>	yes	yes	
<code>overlayXY</code>	no	no	
<code>Pair</code>	partial	no	Only the <code>normal</code> style in a <code>StyleMap</code> is supported. <code>highlight</code> isn't supported.
<code>phoneNumber</code>	yes	yes	
<code>PhotoOverlay</code>	no	no	
<code>Placemark</code>	yes	yes	
<code>Point</code>	yes	yes	

ELEMENT NAME	READ	WRITE	NOTES
Polygon	yes	yes	
PolyStyle	yes	yes	
Region	partial	partial	<code>LatLongBox</code> is supported at the document level.
rotation	no	no	
rotationXY	no	no	
scale	no	no	
Schema	yes	yes	
SchemaData	yes	yes	
schemaUrl	partial	yes	Doesn't support loading styles from external documents that aren't included in a KMZ.
ScreenOverlay	no	no	
screenXY	no	no	
SimpleData	yes	yes	
SimpleField	yes	yes	
size	no	no	
Snippet	partial	partial	<code>maxLines</code> attribute is ignored.
south	yes	yes	
Style	yes	yes	
StyleMap	partial	no	Only the normal style in a <code>StyleMap</code> is supported.
styleUrl	partial	yes	External style URLs aren't supported.
text	yes	yes	Replacement of <code>[\$geDirections]</code> isn't supported
textColor	yes	yes	

ELEMENT NAME	READ	WRITE	NOTES
TimeSpan	yes	yes	
TimeStamp	yes	yes	
value	yes	yes	
viewRefreshMode	partial	no	If pointing to a WMS service, then only <code>onStop</code> is supported for ground overlays. Will append <code>BBOX=[bboxWest], [bboxSouth], [bboxEast], [bboxNorth]</code> to the URL and update as the map moves.
visibility	yes	yes	
west	yes	yes	
when	yes	yes	
width	yes	yes	

GeoRSS elements

The spatial IO module supports the following GeoRSS elements.

ELEMENT NAME	READ	WRITE	NOTES
atom:author	yes	yes	
atom:category	yes	yes	
atom:content	yes	yes	
atom:contributor	yes	yes	
atom:email	yes	yes	
atom:entry	yes	yes	
atom:feed	yes	yes	
atom:icon	yes	yes	
atom:id	yes	yes	
atom:link	yes	yes	
atom:logo	yes	yes	

ELEMENT NAME	READ	WRITE	NOTES
<code>atom:name</code>	yes	yes	
<code>atom:published</code>	yes	yes	
<code>atom:rights</code>	yes	yes	
<code>atom:source</code>	yes	yes	
<code>atom:subtitle</code>	yes	yes	
<code>atom:summary</code>	yes	yes	
<code>atom:title</code>	yes	yes	
<code>atom:updated</code>	yes	yes	
<code>atom:uri</code>	yes	yes	
<code>geo:lat</code>	yes	no	Written as a <code>georss:point</code> .
<code>geo:lon</code>	yes	no	Written as a <code>georss:point</code> .
<code>geo:long</code>	yes	no	Written as a <code>georss:point</code> .
<code>georss:box</code>	yes	no	Read as a polygon and given a <code>subType</code> property of "Rectangle"
<code>georss:circle</code>	yes	yes	
<code>georss:elev</code>	yes	yes	
<code>georss:featurename</code>	yes	yes	
<code>georss:featuretypetag</code>	yes	yes	
<code>georss:floor</code>	yes	yes	
<code>georss:line</code>	yes	yes	
<code>georss:point</code>	yes	yes	
<code>georss:polygon</code>	yes	yes	
<code>georss:radius</code>	yes	yes	

ELEMENT NAME	READ	WRITE	NOTES
<code>georss:relationshiptag</code>	yes	yes	
<code>georss:where</code>	yes	yes	
<code>geourl:latitude</code>	yes	no	Written as a <code>georss:point</code> .
<code>geourl:longitude</code>	yes	no	Written as a <code>georss:point</code> .
<code>position</code>	yes	no	Some XML feeds will wrap GML with a position tag instead of wrapping it with a <code>georss:where</code> tag. Will read this tag, but will write using a <code>georss:where</code> tag.
<code>rss</code>	yes	no	GeoRSS written in ATOM format.
<code>rss:author</code>	yes	partial	Written as an <code>atom:author</code> .
<code>rss:category</code>	yes	partial	Written as an <code>atom:category</code> .
<code>rss:channel</code>	yes	no	
<code>rss:cloud</code>	yes	no	
<code>rss:comments</code>	yes	no	
<code>rss:copyright</code>	yes	partial	Written as an <code>atom:rights</code> if shape doesn't have a <code>rights</code> properties property already.
<code>rss:description</code>	yes	partial	Written as an <code>atom:content</code> if shape doesn't have a <code>content</code> properties property already.
<code>rss:docs</code>	yes	no	
<code>rss:enclosure</code>	yes	no	
<code>rss:generator</code>	yes	no	

ELEMENT NAME	READ	WRITE	NOTES
<code>rss:guid</code>	yes	partial	Written as an <code>atom:id</code> if shape doesn't have an <code>id</code> <code>properties</code> property already.
<code>rss:image</code>	yes	partial	Written as an <code>atom:logo</code> if shape doesn't have a <code>logo</code> <code>properties</code> property already.
<code>rss:item</code>	yes	partial	Written as an <code>atom:entry</code> .
<code>rss:language</code>	yes	no	
<code>rss:lastBuildDate</code>	yes	partial	Written as an <code>atom:updated</code> if shape doesn't have an <code>updated</code> <code>properties</code> property already.
<code>rss:link</code>	yes	partial	Written as an <code>atom:link</code> .
<code>rss:managingEditor</code>	yes	partial	Written as an <code>atom:contributor</code> .
<code>rss:pubDate</code>	yes	partial	Written as an <code>atom:published</code> if shape doesn't have a <code>published</code> <code>properties</code> property already.
<code>rss:rating</code>	yes	no	
<code>rss:skipDays</code>	yes	no	
<code>rss:skipHours</code>	yes	no	
<code>rss:source</code>	yes	partial	Written as an <code>atom:source</code> containing an <code>atom:link</code> .
<code>rss:textInput</code>	yes	no	
<code>rss:title</code>	yes	partial	Written as an <code>atom:title</code> .
<code>rss:ttl</code>	yes	no	
<code>rss:webMaster</code>	yes	no	

GML elements

The spatial IO module supports the following GML elements.

ELEMENT NAME	READ	WRITE	NOTES
<code>gml:coordinates</code>	yes	no	Written as <code>gml:posList</code> .
<code>gml:curveMember</code>	yes	no	
<code>gml:curveMembers</code>	yes	no	
<code>gml:Box</code>	yes	no	Written as <code>gml:Envelope</code> .
<code>gml:description</code>	yes	yes	
<code>gml:Envelope</code>	yes	yes	
<code>gml:exterior</code>	yes	yes	
<code>gml:Feature</code>	yes	no	Written as a shape.
<code>gml:FeatureCollection</code>	yes	no	Written as a geometry collection.
<code>gml:featureMember</code>	yes	no	Written as a geometry collection.
<code>gml:geometry</code>	yes	no	Written as a shape.
<code>gml:geometryMember</code>	yes	yes	
<code>gml:geometryMembers</code>	yes	yes	
<code>gml:identifier</code>	yes	yes	
<code>gml:innerBoundaryIs</code>	yes	no	Written using <code>gml.interior</code> .
<code>gml:interior</code>	yes	yes	
<code>gml:LinearRing</code>	yes	yes	
<code>gml:LineString</code>	yes	yes	
<code>gml:lineStringMember</code>	yes	yes	
<code>gml:lineStringMembers</code>	yes	no	
<code>gml:MultiCurve</code>	yes	no	Only reads <code>gml:LineString</code> members. Written as <code>gml.MultiLineString</code>
<code>gml:MultiGeometry</code>	partial	partial	Only read as a FeatureCollection.

ELEMENT NAME	READ	WRITE	NOTES
gml:MultiLineString	yes	yes	
gml:MultiPoint	yes	yes	
gml:MultiPolygon	yes	yes	
gml:MultiSurface	yes	no	Only reads gml:Polygon members. Written as gml.MultiPolygon
gml:name	yes	yes	
gml:outerBoundaryIs	yes	no	Written using gml.exterior .
gml:Point	yes	yes	
gml:pointMember	yes	yes	
gml:pointMembers	yes	no	
gml:Polygon	yes	yes	
gml:polygonMember	yes	yes	
gml:polygonMembers	yes	no	
gml:pos	yes	yes	
gml:posList	yes	yes	
gml:surfaceMember	yes	yes	

additional notes

- Member elements will be searched for a geometry that may be buried within child elements. This search operation is necessary as many XML formats that extend from GML may not place a geometry as a direct child of a member element.
- srsName is partially supported for WGS84 coordinates and the following codes:[EPSG:4326](#), and web Mercator ([EPSG:3857](#) or one of its alternative codes. Any other coordinate system will be parsed as WGS84 as-is.
- Unless specified when reading an XML feed, the axis order is determined based on hints in the XML feed. A preference is given for the "latitude, longitude" axis order.
- Unless a custom GML namespace is specified for the properties when writing to a GML file, additional property information will not be added.

GPX elements

The spatial IO module supports the following GPX elements.

ELEMENT NAME	READ	WRITE	NOTES
gpx:ageofdgpsdata	yes	yes	
gpx:author	yes	yes	
gpx:bounds	yes	yes	Converted into a LocationRect when read.
gpx:cmt	yes	yes	
gpx:copyright	yes	yes	
gpx:desc	yes	yes	Copied into a description property when read to align with other XML formats.
gpx:dgpsid	yes	yes	
gpx:ele	yes	yes	
gpx:extensions	partial	partial	When read, style information is extracted. All other extensions will be flattened into a simple JSON object. Only shape style information is written.
gpx:geoidheight	yes	yes	
gpx:gpx	yes	yes	
gpx:hdop	yes	yes	
gpx:link	yes	yes	
gpx:magvar	yes	yes	
gpx:metadata	yes	yes	
gpx:name	yes	yes	
gpx:pdop	yes	yes	
gpx:rte	yes	yes	
gpx:rtept	yes	yes	
gpx:sat	yes	yes	
gpx:src	yes	yes	

ELEMENT NAME	READ	WRITE	NOTES
<code>gpx: sym</code>	yes	yes	Value is captured, but it isn't used to alter the pushpin icon.
<code>gpx: text</code>	yes	yes	
<code>gpx: time</code>	yes	yes	
<code>gpx: trk</code>	yes	yes	
<code>gpx: trkpt</code>	yes	yes	
<code>gpx: trkseg</code>	yes	yes	
<code>gpx: type</code>	yes	yes	
<code>gpx: vdop</code>	yes	yes	
<code>gpx: wpt</code>	yes	yes	
<code>gpx_style:color</code>	yes	yes	
<code>gpx_style:line</code>	partial	partial	<code>color</code> , <code>opacity</code> , <code>width</code> , <code>lineCap</code> are supported.
<code>gpx_style:opacity</code>	yes	yes	
<code>gpx_style:width</code>	yes	yes	
<code>gpxx:DisplayColor</code>	yes	no	Used to specify the color of a shape. When writing, <code>gpx_style:line</code> color will be used instead.
<code>gpxx:RouteExtension</code>	partial	no	All properties are read into <code>properties</code> . Only <code>DisplayColor</code> is used.
<code>gpxx:TrackExtension</code>	partial	no	All properties are read into <code>properties</code> . Only <code>DisplayColor</code> is used.
<code>gpxx:WaypointExtension</code>	partial	no	All properties are read into <code>properties</code> . Only <code>DisplayColor</code> is used.
<code>gpx: keywords</code>	yes	yes	
<code>gpx: fix</code>	yes	yes	

additional notes

When writing;

- MultiPoints will be broken up into individual waypoints.
- Polygons and MultiPolygons will be written as tracks.

Supported Well-Known Text geometry types

GEOMETRY TYPE	READ	WRITE
POINT	x	x
POINT Z	x	x
POINT M	x	x ^[2]
POINT ZM	x ^{[1][2]}	
LINESTRING	x	x
LINESTRING Z	x	x
LINESTRING M	x	x ^[2]
LINESTRING ZM	x ^{[1][2]}	
POLYGON	x	x
POLYGON Z	x	x
POLYGON M	x	x ^[2]
POLYGON ZM	x ^{[1][2]}	
MULTIPOINT	x	x
MULTIPOINT Z	x	x
MULTIPOINT M	x	x ^[2]
POLYMULTIPOINT ZM	x ^{[1][2]}	
MULTILINESTRING	x	x
MULTILINESTRING Z	x	x
MULTILINESTRING M	x	x ^[2]
MULTILINESTRING ZM	x ^{[1][2]}	
MULTIPOLYGON	x	x
MULTIPOLYGON Z	x	x

GEOMETRY TYPE	READ	WRITE
MULTIPOLYGON M	x	x ^[2]
MULTIPOLYGON ZM	x ^{[1][2]}	
GEOMETRYCOLLECTION	x	x
GEOMETRYCOLLECTION Z	x	x
GEOMETRYCOLLECTION M	x	x ^[2]
GEOMETRYCOLLECTION ZM	x ^{[1][2]}	x

[1] Only Z parameter is captured and added as a third value in the Position value.

[2] M parameter isn't captured.

Delimited spatial data support

Delimited spatial data, such as comma-separated value files (CSV), often have columns that contain spatial data. For example, there could be columns that contain latitude and longitude information. In Well-Known Text format, there could be a column that contains spatial geometry data.

Spatial data column detection

When reading a delimited file that contains spatial data, the header will be analyzed to determine which columns contain location fields. If the header contains type information, it will be used to cast the cell values to the appropriate type. If no header is specified, the first row will be analyzed and used to generate a header. When analyzing the first row, a check is executed to match column names with the following names in a case-insensitive way. The order of the names is the priority, in case two or more names exist in a file.

Latitude

- `latitude`
- `lat`
- `latdd`
- `lat_dd`
- `latitude83`
- `latdecdeg`
- `y`
- `ycenter`
- `point-y`

Longitude

- `longitude`
- `lon`
- `lng`
- `long`
- `longdd`
- `long_dd`
- `longitude83`
- `longdecdeg`

- `x`
- `xcenter`
- `point-x`

Elevation

- `elevation`
- `elv`
- `altitude`
- `alt`
- `z`

Geography

The first row of data will be scanned for strings that are in Well-Known Text format.

Delimited data column types

When scanning the header row, any type information that is in the column name will be extracted and used to cast the cells in that column. Here is an example of a column name that has a type value: "ColumnName (typeName)". The following case-insensitive type names are supported:

Numbers

- `edm.int64`
- `int`
- `long`
- `edm.double`
- `float`
- `double`
- `number`

Booleans

- `edm.boolean`
- `bool`
- `boolean`

Dates

- `edm.datetime`
- `date`
- `datetime`

Geography

- `edm.geography`
- `geography`

Strings

- `edm.string`
- `varchar`
- `text`
- `string`

If no type information can be extracted from the header, and the dynamic typing option is enabled when reading, then each cell will be individually analyzed to determine what data type it is best suited to be cast as.

Next steps

See the following articles for more code samples to add to your maps:

[Read and write spatial data](#)

Web SDK supported browsers

3/5/2021 • 2 minutes to read • [Edit Online](#)

The Azure Maps Web SDK provides a helper function called [atlas.isSupported](#). This function detects whether a web browser has the minimum set of WebGL features required to support loading and rendering the map control. Here's an example of how to use the function:

```
if (!atlas.isSupported()) {
    alert('Your browser is not supported by Azure Maps');
} else if (!atlas.isSupported(true)) {
    alert('Your browser is supported by Azure Maps, but may have major performance caveats.');
} else {
    // Your browser is supported. Add your map code here.
}
```

Desktop

The Azure Maps Web SDK supports the following desktop browsers:

- Microsoft Edge (current and previous version)
- Google Chrome (current and previous version)
- Mozilla Firefox (current and previous version)
- Apple Safari (macOS X) (current and previous version)

See also [Target legacy browsers](#) later in this article.

Mobile

The Azure Maps Web SDK supports the following mobile browsers:

- Android
 - Current version of Chrome on Android 6.0 and later
 - Chrome WebView on Android 6.0 and later
- iOS
 - Mobile Safari on the current and previous major version of iOS
 - UIWebView and WKWebView on the current and previous major version of iOS
 - Current version of Chrome for iOS

TIP

If you're embedding a map inside a mobile application by using a `WebView` control, you might prefer to use the [npm package of the Azure Maps Web SDK](#) instead of referencing the version of the SDK that's hosted on Azure Content Delivery Network. This approach reduces loading time because the SDK is already be on the user's device and doesn't need to be downloaded at run time.

Node.js

The following Web SDK modules are also supported in Node.js:

- Services module ([documentation](#) | [npm module](#))

Target legacy browsers

You might want to target older browsers that don't support WebGL or that have only limited support for it. In such cases, we recommend that you use Azure Maps services together with an open-source map control like [Leaflet](#). Here's an example that makes use of the open source [Azure Maps Leaflet plugin](#).

<https://codepen.io/azuremaps/embed/GeLgyx/?height=500&theme-id=0&default-tab=html,result&rerun-position=hidden&>

Additional code samples using Azure Maps in Leaflet can be found [here](#).

[Here](#) are some popular open-source map controls that the Azure Maps team has created plugin's for.

Next steps

Learn more about the Azure Maps Web SDK:

[Map control](#)

[Services module](#)

Azure Maps Web SDK best practices

4/27/2021 • 30 minutes to read • [Edit Online](#)

This document focuses on best practices for the Azure Maps Web SDK, however, many of the best practices and optimizations outlined can be applied to all other Azure Maps SDKs.

The Azure Maps Web SDK provides a powerful canvas for rendering large spatial data sets in many different ways. In some cases, there are multiple ways to render data the same way, but depending on the size of the data set and the desired functionality, one method may perform better than others. This article highlights best practices and tips and tricks to maximize performance and create a smooth user experience.

Generally, when looking to improve performance of the map, look for ways to reduce the number of layers and sources, and the complexity of the data sets and rendering styles being used.

Security basics

The single most important part of your application is its security. If your application isn't secure a hacker can ruin any application, no matter how good the user experience might be. The following are some tips to keep your Azure Maps application secure. When using Azure, be sure to familiarize yourself with the security tools available to you. See this document for an [introduction to Azure security](#).

IMPORTANT

Azure Maps provides two methods of authentication.

- Subscription key-based authentication
- Azure Active Directory authentication Use Azure Active Directory in all production applications. Subscription key-based authentication is simple and what most mapping platforms use as a light way method to measure your usage of the platform for billing purposes. However, this is not a secure form of authentication and should only be used locally when developing apps. Some platforms provide the ability to restrict which IP addresses and/or HTTP referrer is in requests, however, this information can easily be spoofed. If you do use subscription keys, be sure to [rotate them regularly](#). Azure Active Directory is an enterprise identity service that has a large selection of security features and settings for all sorts of application scenarios. Microsoft recommends that all production applications using Azure Maps use Azure Active Directory for authentication. Learn more about [managing authentication in Azure Maps](#) in this document.

Secure your private data

When data is added to the Azure Maps interactive map SDKs, it is rendered locally on the end user's device and is never sent back out to the internet for any reason.

If your application is loading data that should not be publicly accessible, make sure that the data is stored in a secure location, is accessed in a secure manner, and that the application itself is locked down and only available to your desired users. If any of these steps are skipped, an unauthorized person has the potential to access this data. Azure Active Directory can assist you with locking this down.

See this tutorial on [adding authentication to your web app running on Azure App Service](#)

Use the latest versions of Azure Maps

The Azure Maps SDKs go through regular security testing along with any external dependency libraries that may be used by the SDKs. Any known security issue is fixed in a timely manner and released to production. If your application points to the latest major version of the hosted version of the Azure Maps Web SDK, it will

automatically receive all minor version updates that will include security related fixes.

If self-hosting the Azure Maps Web SDK via the NPM module, be sure to use the caret (^) symbol to in combination with the Azure Maps NPM package version number in your `package.json` file so that it will always point to the latest minor version.

```
"dependencies": {  
    "azure-maps-control": "^2.0.30"  
}
```

Optimize initial map load

When a web page is loading, one of the first things you want to do is start rendering something as soon as possible so that the user isn't staring at a blank screen.

Watch the maps ready event

Similarly, when the map initially loads often it is desired to load data on it as quickly as possible, so the user isn't looking at an empty map. Since the map loads resources asynchronously, you have to wait until the map is ready to be interacted with before trying to render your own data on it. There are two events you can wait for, a `load` event and a `ready` event. The load event will fire after the map has finished completely loading the initial map view and every map tile has loaded. The ready event will fire when the minimal map resources needed to start interacting with the map. The ready event can often fire in half the time of the load event and thus allow you to start loading your data into the map sooner.

Lazy load the Azure Maps Web SDK

If the map isn't needed right away, lazy load the Azure Maps Web SDK until it is needed. This will delay the loading of the JavaScript and CSS files used by the Azure Maps Web SDK until needed. A common scenario where this occurs is when the map is loaded in a tab or flyout panel that isn't displayed on page load. The following code sample shows how to delay the loading the Azure Maps Web SDK until a button is pressed.

<https://codepen.io/azuremaps/embed/vYEeyOv?height=500&theme-id=default&default-tab=js,result&rerun-position=hidden&>

Add a placeholder for the map

If the map takes a while to load due to network limitations or other priorities within your application, consider adding a small background image to the map `div` as a placeholder for the map. This will fill the void of the map `div` while it is loading.

Set initial map style and camera options on initialization

Often apps want to load the map to a specific location or style. Sometimes developers will wait until the map has loaded (or wait for the `ready` event), and then use the `setCenter` or `setStyle` functions of the map. This will often take longer to get to the desired initial map view since many resources end up being loaded by default before the resources needed for the desired map view are loaded. A better approach is to pass in the desired map camera and style options into the map when initializing it.

Optimize data sources

The Web SDK has two data sources,

- **GeoJSON source:** Known as the `DataSource` class, manages raw location data in GeoJSON format locally. Good for small to medium data sets (upwards of hundreds of thousands of features).
- **Vector tile source:** Known as the `VectorTileSource` class, loads data formatted as vector tiles for the current map view, based on the maps tiling system. Ideal for large to massive data sets (millions or billions of

features).

Use tile-based solutions for large datasets

If working with larger datasets containing millions of features, the recommended way to achieve optimal performance is to expose the data using a server-side solution such as vector or raster image tile service. Vector tiles are optimized to load only the data that is in view with the geometries clipped to the focus area of the tile and generalized to match the resolution of the map for the zoom level of the tile.

The [Azure Maps Creator platform](#) provides the ability to retrieve data in vector tile format. Other data formats can be used using tools such as [Tippecanoe](#) or one of the many [resources list on this page](#).

It is also possible to create a custom service that renders datasets as raster image tiles on the server-side and load the data using the `TileLayer` class in the map SDK. This provides exceptional performance as the map only needs to load and manage a few dozen images at most. However, there are some limitations with using raster tiles since the raw data is not available locally. A secondary service is often required to power any type of interaction experience, for example, find out what shape a user clicked on. Additionally, the file size of a raster tile is often larger than a compressed vector tile that contains generalized and zoom level optimized geometries.

Learn more about data sources in the [Create a data source](#) document.

Combine multiple datasets into a single vector tile source

The less data sources the map has to manage, the faster it can process all features to be displayed. In particular, when it comes to tile sources, combining two vector tile sources together cuts the number of HTTP requests to retrieve the tiles in half, and the total amount of data would be slightly smaller since there is only one file header.

Combining multiple data sets in a single vector tile source can be achieved using a tool such as [Tippecanoe](#). Data sets can be combined into a single feature collection or separated into separate layers within the vector tile known as source-layers. When connecting a vector tile source to a rendering layer, you would specify the source-layer that contains the data that you want to render with the layer.

Reduce the number of canvas refreshes due to data updates

There are several ways data in a `DataSource` class can be added or updated. Listed below are the different methods and some considerations to ensure good performance.

- The data sources `add` function can be used to add one or more features to a data source. Each time this function is called it will trigger a map canvas refresh. If adding many features, combine them into an array or feature collection and passing them into this function once, rather than looping over a data set and calling this function for each feature.
- The data sources `setShapes` function can be used to overwrite all shapes in a data source. Under the hood, it combines the data sources `clear` and `add` functions together and does a single map canvas refresh instead of two, which is much faster. Be sure to use this when you want to update all data in a data source.
- The data sources `importDataFromUrl` function can be used to load a GeoJSON file via a URL into a data source. Once the data has been downloaded, it is passed into the data sources `add` function. If the GeoJSON file is hosted on a different domain, be sure that the other domain supports cross domain requests (CORS). If it doesn't consider copying the data to a local file on your domain or creating a proxy service that has CORS enabled. If the file is large, consider converting it into a vector tile source.
- If features are wrapped with the `Shape` class, the `addProperty`, `setCoordinates`, and `setProperties` functions of the shape will all trigger an update in the data source and a map canvas refresh. All features returned by the data sources `getShapes` and `getShapeById` functions are automatically wrapped with the `Shape` class. If you want to update several shapes, it is faster to convert them to JSON using the data sources `toJson` function, editing the GeoJSON, then passing this data into the data sources `setShapes` function.

Avoid calling the data sources clear function unnecessarily

Calling the `clear` function of the `DataSource` class causes a map canvas refresh. If the `clear` function is called

multiple times in a row, a delay can occur while the map waits for each refresh to occur.

A common scenario where this often appears in applications is when an app clears the data source, downloads new data, clears the data source again then adds the new data to the data source. Depending on the desired user experience, the following alternatives would be better.

- Clear the data before downloading the new data, then pass the new data into the data sources `add` or `setShapes` function. If this is the only data set on the map, the map will be empty while the new data is downloading.
- Download the new data, then pass it into the data sources `setShapes` function. This will replace all the data on the map.

Remove unused features and properties

If your dataset contains features that aren't going to be used in your app, remove them. Similarly, remove any properties on features that aren't needed. This has several benefits:

- Reduces the amount of data that has to be downloaded.
- Reduces the number of features that need to be looped through when rendering the data.
- Can sometimes help simplify or remove data-driven expressions and filters, which mean less processing required at render time.

When features have a lot of properties or content, it is much more performant to limit what gets added to the data source to just those needed for rendering and to have a separate method or service for retrieving the additional property or content when needed. For example, if you have a simple map displaying locations on a map when clicked a bunch of detailed content is displayed. If you want to use data driven styling to customize how the locations are rendered on the map, only load the properties needed into the data source. When you want to display the detailed content, use the ID of the feature to retrieve the additional content separately. If the content is stored on the server-side, a service can be used to retrieve it asynchronously, which would drastically reduce the amount of data that needs to be downloaded when the map is initially loaded.

Additionally, reducing the number of significant digits in the coordinates of features can also significantly reduce the data size. It is not uncommon for coordinates to contain 12 or more decimal places; however, six decimal places have an accuracy of about 0.1 meter, which is often more precise than the location the coordinate represents (six decimal places is recommended when working with small location data such as indoor building layouts). Having any more than six decimal places will likely make no difference in how the data is rendered and will only require the user to download more data for no added benefit.

Here is a list of [useful tools for working with GeoJSON data](#).

Use a separate data source for rapidly changing data

Sometimes there is a need to rapidly update data on the map for things such as showing live updates of streaming data or animating features. When a data source is updated, the rendering engine will loop through and render all features in the data source. Separating static data from rapidly changing data into different data sources can significantly reduce the number of features that are re-rendered on each update to the data source and improve overall performance.

If using vector tiles with live data, an easy way to support updates is to use the `expires` response header. By default, any vector tile source or raster tile layer will automatically reload tiles when the `expires` date. The traffic flow and incident tiles in the map use this feature to ensure fresh real-time traffic data is displayed on the map. This feature can be disabled by setting the maps `refreshExpiredTiles` service option to `false`.

Adjust the buffer and tolerance options in GeoJSON data sources

The `DataSource` class converts raw location data into vector tiles local for on-the-fly rendering. These local vector tiles clip the raw data to the bounds of the tile area with a bit of buffer to ensure smooth rendering between tiles. The smaller the `buffer` option is, the fewer overlapping data is stored in the local vector tiles and

the better performance, however, the greater the chance of rendering artifacts occurring. Try tweaking this option to get the right mix of performance with minimal rendering artifacts.

The `DataSource` class also has a `tolerance` option that is used with the Douglas-Peucker simplification algorithm when reducing the resolution of geometries for rendering purposes. Increasing this tolerance value will reduce the resolution of geometries and in turn improve performance. Tweak this option to get the right mix of geometry resolution and performance for your data set.

Set the max zoom option of GeoJSON data sources

The `DataSource` class converts raw location data into vector tiles local for on-the-fly rendering. By default, it will do this until zoom level 18, at which point, when zoomed in closer, it will sample data from the tiles generated for zoom level 18. This works well for most data sets that need to have high resolution when zoomed in at these levels. However, when working with data sets that are more likely to be viewed when zoomed out more, such as when viewing state or province polygons, setting the `minZoom` option of the data source to a smaller value such as `12` will reduce the amount computation, local tile generation that occurs, and memory used by the data source and increase performance.

Minimize GeoJSON response

When loading GeoJSON data from a server either through a service or by loading a flat file, be sure to have the data minimized to remove unneeded space characters that makes the download size larger than needed.

Access raw GeoJSON using a URL

It is possible to store GeoJSON objects inline inside of JavaScript, however this will use a lot of memory as copies of it will be stored across the variable you created for this object and the data source instance, which manages it within a separate web worker. Expose the GeoJSON to your app using a URL instead and the data source will load a single copy of data directly into the data sources web worker.

Optimize rendering layers

Azure maps provides several different layers for rendering data on a map. There are many optimizations you can take advantage of to tailor these layers to your scenario the increase performances and the overall user experience.

Create layers once and reuse them

The Azure Maps Web SDK is decided to be data driven. Data goes into data sources, which are then connected to rendering layers. If you want to change the data on the map, update the data in the data source or change the style options on a layer. This is often much faster than removing and then recreating layers whenever there is a change.

Consider bubble layer over symbol layer

The bubble layer renders points as circles on the map and can easily have their radius and color styled using a data-driven expression. Since the circle is a simple shape for WebGL to draw, the rendering engine will be able to render these much faster than a symbol layer, which has to load and render an image. The performance difference of these two rendering layers is noticeable when rendering tens of thousands of points.

Use HTML markers and Popups sparingly

Unlike most layers in the Azure Maps Web control that use WebGL for rendering, HTML Markers and Popups use traditional DOM elements for rendering. As such, the more HTML markers and Popups added a page, the more DOM elements there are. Performance can degrade after adding a few hundred HTML markers or popups. For larger data sets, consider either clustering your data or using a symbol or bubble layer. For popups, a common strategy is to create a single popup and reuse it by updating its content and position as shown in the below example:

<https://codepen.io/azuremaps/embed/rQbjvK?height=500&theme-id=0&default-tab=js,result&embed-version=2&editable=true&rerun-position=hidden&>

That said, if you only have a few points to render on the map, the simplicity of HTML markers may be preferred. Additionally, HTML markers can easily be made draggable if needed.

Combine layers

The map is capable of rendering hundreds of layers, however, the more layers there are, the more time it takes to render a scene. One strategy to reduce the number of layers is to combine layers that have similar styles or can be styled using a [data-driven styles](#).

For example, consider a data set where all features have a `isHealthy` property that can have a value of `true` or `false`. If creating a bubble layer that renders different colored bubbles based on this property, there are several ways to do this as listed below from least performant to most performant.

- Split the data into two data sources based on the `isHealthy` value and attach a bubble layer with a hard-coded color option to each data source.
- Put all the data into a single data source and create two bubble layers with a hard-coded color option and a filter based on the `isHealthy` property.
- Put all the data into a single data source, create a single bubble layer with a `case` style expression for the color option based on the `isHealthy` property. Here is a code sample that demonstrates this.

```
var layer = new atlas.layer.BubbleLayer(source, null, {
  color: [
    'case',
    //Get the 'isHealthy' property from the feature.
    ['get', 'isHealthy'],
    //If true, make the color 'green'.
    'green',
    //If false, make the color red.
    'red'
  ]
});
```

Create smooth symbol layer animations

Symbol layers have collision detection enabled by default. This collision detection aims to ensure that no two symbols overlap. The icon and text options of a symbol layer have two options,

- `allowOverlap` - specifies if the symbol will be visible if it collides with other symbols.
- `ignorePlacement` - specifies if the other symbols are allowed to collide with the symbol.

Both of these options are set to `false` by default. When animating a symbol, the collision detection calculations will run on each frame of the animation, which can slow down the animation and make it look less fluid. To smooth the animation out, set these options to `true`.

The following code sample shows a simple way to animate a symbol layer.

<https://codepen.io/azuremaps/embed/oNgGzRd?height=500&theme-id=default&default-tab=js,result&rerun-position=hidden&>

Specify zoom level range

If your data meets one of the following criteria, be sure to specify the min and max zoom level of the layer so that the rendering engine can skip it when outside of the zoom level range.

- If the data is coming from a vector tile source, often source layers for different data types are only available through a range of zoom levels.
- If using a tile layer that doesn't have tiles for all zoom levels 0 through 24 and you want it to only render at the levels it has tiles, and not try to fill in missing tiles with tiles from other zoom levels.
- If you only want to render a layer at certain zoom levels. All layers have a `minZoom` and `maxZoom` option where the layer will be rendered when between these zoom levels based on this logic
`maxZoom > zoom >= minZoom`.

Example

```
//Only render this layer between zoom levels 1 and 9.
var layer = new atlas.layer.BubbleLayer(dataSource, null, {
  minZoom: 1,
  maxZoom: 10
});
```

Specify tile layer bounds and source zoom range

By default, tile layers will load tiles across the whole globe. However, if the tile service only has tiles for a certain area the map will try to load tiles outside of this area. When this happens, a request for each tile will be made and wait for a response that can block other requests being made by the map and thus slow down the rendering of other layers. Specifying the bounds of a tile layer will result in the map only requesting tiles that are within that bounding box. Also, if the tile layer is only available between certain zoom levels, specify the min and max source zoom for the same reason.

Example

```
var tileLayer = new atlas.layer.TileLayer({
  tileSize: 'myTileServer/{z}/{y}/{x}.png',
  bounds: [-101.065, 14.01, -80.538, 35.176],
  minSourceZoom: 1,
  maxSourceZoom: 10
});
```

Use a blank map style when base map not visible

If a layer is being overlaid on the map that will completely cover the base map, consider setting the map style to `blank` or `blank_accessible` so that the base map isn't rendered. A common scenario for doing this is when overlaying a full globe tile that has no opacity or transparent area above the base map.

Smoothly animate image or tile layers

If you want to animate through a series of image or tile layers on the map. It is often faster to create a layer for each image or tile layer and to change the opacity than to update the source of a single layer on each animation frame. Hiding a layer by setting the opacity to zero and showing a new layer by setting its opacity to a value greater than zero is much faster than updating the source in the layer. Alternatively, the visibility of the layers can be toggled, but be sure to set the fade duration of the layer to zero, otherwise it will animate the layer when displaying it, which will cause a flicker effect since the previous layer would have been hidden before the new layer is visible.

Tweak Symbol layer collision detection logic

The symbol layer has two options that exist for both icon and text called `allowOverlap` and `ignorePlacement`. These two options specify if the icon or text of a symbol can overlap or be overlapped. When these are set to `false`, the symbol layer will do calculations when rendering each point to see if it collides with any other already rendered symbol in the layer, and if it does, will not render the colliding symbol. This is good at reducing clutter on the map and reducing the number of objects rendered. By setting these options to `false`, this collision detection logic will be skipped, and all symbols will be rendered on the map. Tweak this option to get

the best combination of performance and user experience.

Cluster large point data sets

When working with large sets of data points you may find that when rendered at certain zoom levels, many of the points overlap and are only partially visible, if at all. Clustering is the process of grouping points that are close together and representing them as a single clustered point. As the user zooms the map in, clusters will break apart into their individual points. This can significantly reduce the amount of data that needs to be rendered, make the map feel less cluttered, and improve performance. The `DataSource` class has options for clustering data locally. Additionally, many tools that generate vector tiles also have clustering options.

Additionally, increase the size of the cluster radius to improve performance. The larger the cluster radius, the less clustered points there is to keep track of and render. Learn more in the [Clustering point data document](#)

Use weighted clustered heat maps

The heat map layer can render tens of thousands of data points easily. For larger data sets, consider enabling clustering on the data source and using a small cluster radius and use the clusters `point_count` property as a weight for the height map. When the cluster radius is only a few pixels in size, there will be little visual difference in the rendered heat map. Using a larger cluster radius will improve performance more but may reduce the resolution of the rendered heat map.

```
var layer = new atlas.layer.HeatMapLayer(source, null, {
  weight: ['get', 'point_count']
});
```

Learn more in the [Clustering and heat maps in this document](clustering-point-data-web-sdk.md #clustering-and-the-heat-maps-layer)

Keep image resources small

Images can be added to the map's image sprite for rendering icons in a symbol layer or patterns in a polygon layer. Keep these images small to minimize the amount of data that has to be downloaded and the amount of space they take up in the map's image sprite. When using a symbol layer that scales the icon using the `size` option, use an image that is the maximum size your plan to display on the map and no bigger. This will ensure the icon is rendered with high resolution while minimizing the resources it uses. Additionally, SVG's can also be used as a smaller file format for simple icon images.

Optimize expressions

[Data-driven style expressions](#) provide a lot of flexibility and power for filtering and styling data on the map.

There are many ways in which expressions can be optimized. Here are a few tips.

Reduce the complexity of filters

Filters loop over all data in a data source and check to see if each filter matches the logic in the filter. If filters become complex, this can cause performance issues. Some possible strategies to address this include the following.

- If using vector tiles, break up the data into different source layers.
- If using the `DataSource` class, break that data up into separate data sources. Try to balance the number of data sources with the complexity of the filter. Too many data sources can cause performance issues too, so you might need to do some testing to find out what works best for your scenario.
- When using a complex filter on a layer, consider using multiple layers with style expressions to reduce the complexity of the filter. Avoid creating a bunch of layers with hardcoded styles when style expressions can be used as a large number of layers can also cause performance issues.

Make sure expressions don't produce errors

Expressions are often used to generate code to perform calculations or logical operations at render time. Just like the code in the rest of your application, be sure the calculations and logical make sense and are not error prone. Errors in expressions will cause issues in evaluating the expression, which can result in reduced performance and rendering issues.

One common error to be mindful of is having an expression that relies on a feature property that might not exist on all features. For example, the following code uses an expression to set the color property of a bubble layer to the `myColor` property of a feature.

```
var layer = new atlas.layer.BubbleLayer(source, null, {
    color: ['get', 'myColor']
});
```

The above code will function fine if all features in the data source have a `myColor` property, and the value of that property is a color. This may not be an issue if you have complete control of the data in the data source and know for certain all features will have a valid color in a `myColor` property. That said, to make this code safe from errors, a `case` expression can be used with the `has` expression to check that the feature has the `myColor` property. If it does, the `to-color` type expression can then be used to try to convert the value of that property to a color. If the color is invalid, a fallback color can be used. The following code demonstrates how to do this and sets the fallback color to green.

```
var layer = new atlas.layer.BubbleLayer(source, null, {
    color: [
        'case',
        //Check to see if the feature has a 'myColor' property.
        ['has', 'myColor'],
        //If true, try validating that 'myColor' value is a color, or fallback to 'green'.
        ['to-color', ['get', 'myColor'], 'green'],
        //If false, return a fallback value.
        'green'
    ]
});
```

Order boolean expressions from most specific to least specific

When using boolean expressions that contain multiple conditional tests, order the conditional tests from most specific to least specific. By doing this, the first condition should reduce the amount of data the second condition has to be tested against, thus reducing the total number of conditional tests that need to be performed.

Simplify expressions

Expressions can be powerful and sometimes complex. The simpler an expression is, the faster it will be evaluated. For example, if a simple comparison is needed, an expression like

`['==', ['get', 'category'], 'restaurant']` would be better than using a match expression like `['match', ['get', 'category'], 'restaurant', true, false]`. In this case, if the property being checked is a boolean value, a `get` expression would be even simpler `['get', 'isRestaurant']`.

Web SDK troubleshooting

The following are some tips to debugging some of the common issues encountered when developing with the Azure Maps Web SDK.

Why doesn't the map display when I load the web control?

Do the following:

- Ensure that you have added your added authentication options to the map. If this is not added, the map will load with a blank canvas since it can't access the base map data without authentication and 401 errors will appear in the network tab of the browser's developer tools.
- Ensure that you have an internet connection.
- Check the console for errors of the browser's developer tools. Some errors may cause the map not to render. Debug your application.
- Ensure you are using a [supported browser](#).

All my data is showing up on the other side of the world, what's going on? Coordinates, also referred to as positions, in the Azure Maps SDKs aligns with the geospatial industry standard format of `[longitude, latitude]`. This same format is also how coordinates are defined in the GeoJSON schema; the core data formatted used within the Azure Maps SDKs. If your data is appearing on the opposite side of the world, it is most likely due to the longitude and latitude values being reversed in your coordinate/position information.

Why are HTML markers appearing in the wrong place in the web control?

Things to check:

- If using custom content for the marker, ensure the `anchor` and `pixelOffset` options are correct. By default, the bottom center of the content is aligned with the position on the map.
- Ensure that the CSS file for Azure Maps has been loaded.
- Inspect the HTML marker DOM element to see if any CSS from your app has appended itself to the marker and is affecting its position.

Why are icons or text in the symbol layer appearing in the wrong place? Check that the `anchor` and the `offset` options are correctly configured to align with the part of your image or text that you want to have aligned with the coordinate on the map. If the symbol is only out of place when the map is rotated, check the `rotationAlignment` option. By default, symbols we will rotate with the maps viewport so that they appear upright to the user. However, depending on your scenario, it may be desirable to lock the symbol to the map's orientation. Set the `rotationAlignment` option to `'map'` to do this. If the symbol is only out of place when the map is pitched/tilted, check the `pitchAlignment` option. By default, symbols we will stay upright with the maps viewport as the map is pitched or tilted. However, depending on your scenario, it may be desirable to lock the symbol to the map's pitch. Set the `pitchAlignment` option to `'map'` to do this.

Why isn't any of my data appearing on the map?

Things to check:

- Check the console in the browser's developer tools for errors.
- Ensure that a data source has been created and added to the map, and that the data source has been connected to a rendering layer that has also been added to the map.
- Add break points in your code and step through it to ensure data is being added to the data source and the data source and layers are being added to the map without any errors occurring.
- Try removing data-driven expressions from your rendering layer. It's possible that one of them may have an error in it that is causing the issue.

Can I use the Azure Maps Web SDK in a sandboxed iframe?

Yes. Note that [Safari has a bug](#) that prevents sandboxed iframes from running web workers, which is requirement of the Azure Maps Web SDK. The solution is to add the `"allow-same-origin"` tag to the `sandbox` property of the iframe.

Get support

The following are the different ways to get support for Azure Maps depending on your issue.

How do I report a data issue or an issue with an address?

Azure Maps has a data feedback tool where data issues can be reported and tracked.

<https://feedback.azuremaps.com/> Each issue submitted generates a unique URL you can use to track the progress of the data issue. The time it takes to resolve a data issue varies depending on the type of issue and how easy it is to verify the change is correct. Once fixed, the render service will see the update in the weekly update, while other services such as geocoding and routing will see the update in the monthly update. Detailed instructions on how to report a data issue is provided in this [document](#).

How do I report a bug in a service or API?

<https://azure.com/support>

Where do I get technical help for Azure Maps?

If related to the Azure Maps visual in Power BI: <https://powerbi.microsoft.com/support/> For all other Azure Maps services: <https://azure.com/support> or the developer forums: <https://docs.microsoft.com/answers/topics/azure-maps.html>

How do I make a feature request?

Make a feature request on our user voice site: <https://feedback.azure.com/forums/909172-azure-maps>

Next steps

See the following articles for more tips on improving the user experience in your application.

[Make your application accessible](#)

Learn more about the terminology used by Azure Maps and the geospatial industry.

[Azure Maps glossary](#)

Getting started with Azure Maps Android SDK

3/26/2021 • 3 minutes to read • [Edit Online](#)

The Azure Maps Android SDK is a vector map library for Android. This article guides you through the processes of installing the Azure Maps Android SDK and loading a map.

Prerequisites

Be sure to complete the steps in the [Quickstart: Create an Android app](#) document.

Localizing the map

The Azure Maps Android SDK provides three different ways of setting the language and regional view of the map. The following code shows how to set the language to French ("fr-FR") and the regional view to "Auto".

The first option is to pass the language and view regional information into the `AzureMaps` class using the static `setLanguage` and `setView` methods globally. This will set the default language and regional view across all Azure Maps controls loaded in your app.

```
static {
    //Alternatively use Azure Active Directory authenticate.
    AzureMaps.setAadProperties("<Your aad clientId>", "<Your aad AppId>", "<Your aad Tenant>");

    //Set your Azure Maps Key.
    //AzureMaps.setSubscriptionKey("<Your Azure Maps Key>");

    //Set the language to be used by Azure Maps.
    AzureMaps.setLanguage("fr-FR");

    //Set the regional view to be used by Azure Maps.
    AzureMaps.setView("Auto");
}
```

```
companion object {
    init {
        //Alternatively use Azure Active Directory authenticate.
        AzureMaps.setAadProperties("<Your aad clientId>", "<Your aad AppId>", "<Your aad Tenant>");

        //Set your Azure Maps Key.
        //AzureMaps.setSubscriptionKey("<Your Azure Maps Key>");

        //Set the language to be used by Azure Maps.
        AzureMaps.setLanguage("fr-FR");

        //Set the regional view to be used by Azure Maps.
        AzureMaps.setView("Auto");
    }
}
```

The second option is to pass the language and view information into the map control XML.

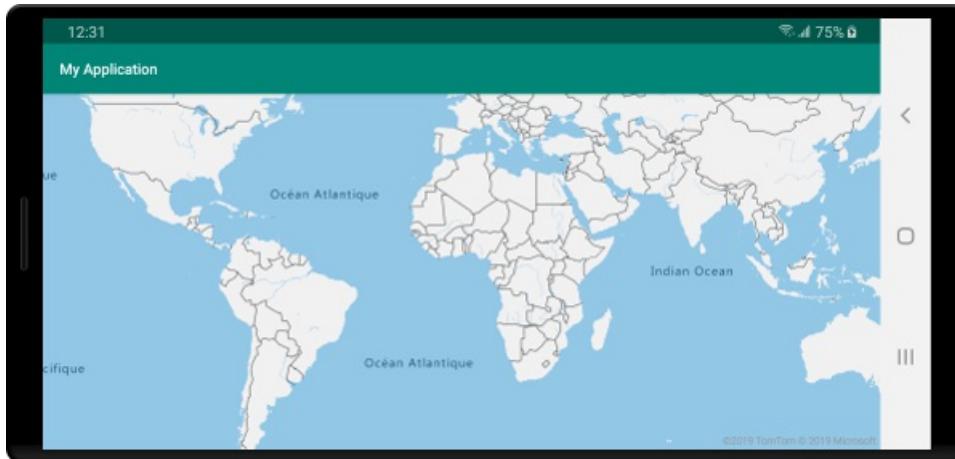
```
<com.microsoft.azure.maps.mapcontrol.MapControl  
    android:id="@+id/myMap"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    app:mapcontrol_language="fr-FR"  
    app:mapcontrol_view="Auto"  
/>
```

The third option is to programmatically set the language and regional view of the map using the maps `setStyle` method. This can be done at any time to change the language and regional view of the map.

```
mapControl.onReady(map -> {  
    map.setStyle(  
        language("fr-FR"),  
        view("Auto")  
    );  
});
```

```
mapControl.onReady(OnReady { map: AzureMap ->  
    map.setStyle(  
        language("fr-FR"),  
        view("Auto")  
    )  
})
```

Here is an example of Azure Maps with the language set to "fr-FR" and regional view set to "Auto".



A complete list of supported languages and regional views is documented [here](#).

Navigating the map

There are several different ways in which the map can be zoomed, panned, rotated, and pitched. The following details all the different ways to navigate the map.

Zoom the map

- Touch the map with two fingers and pinch together to zoom out or spread the fingers apart to zoom in.
- Double tap the map to zoom in one level.
- Double tap with two fingers to zoom out the map one level.
- Tap twice; on second tap, hold your finger on the map and drag up to zoom in, or down to zoom out.

Pan the map

- Touch the map and drag in any direction.

Rotate the map

- Touch the map with two fingers and rotate.

Pitch the map

- Touch the map with two fingers and drag them up or down together.

Azure Government cloud support

The Azure Maps Android SDK supports the Azure Government cloud. The Azure Maps Android SDK is accessed from the same Maven repository. The following tasks will need to be done to connect to the Azure Government cloud version of the Azure Maps platform.

In same place where the Azure Maps authentication details are specified, add the following line of code to tell the map to use the Azure Maps government cloud domain.

```
AzureMaps.setDomain("atlas.azure.us");
```

```
AzureMaps.setDomain("atlas.azure.us")
```

Be sure to use Azure Maps authentication details from the Azure Government cloud platform when authenticating the map and services.

Next steps

Learn how to add overlay data on the map:

[Manage authentication in Azure Maps](#)

[Change map styles in Android maps](#)

[Add a symbol layer](#)

[Add a line layer](#)

[Add a polygon layer](#)

Set map style (Android SDK)

4/30/2021 • 4 minutes to read • [Edit Online](#)

This article shows you two ways to set map styles using the Azure Maps Android SDK. Azure Maps has six different maps styles to choose from. For more information about supported map styles, see [supported map styles in Azure Maps](#).

Prerequisites

Be sure to complete the steps in the [Quickstart: Create an Android app](#) document.

IMPORTANT

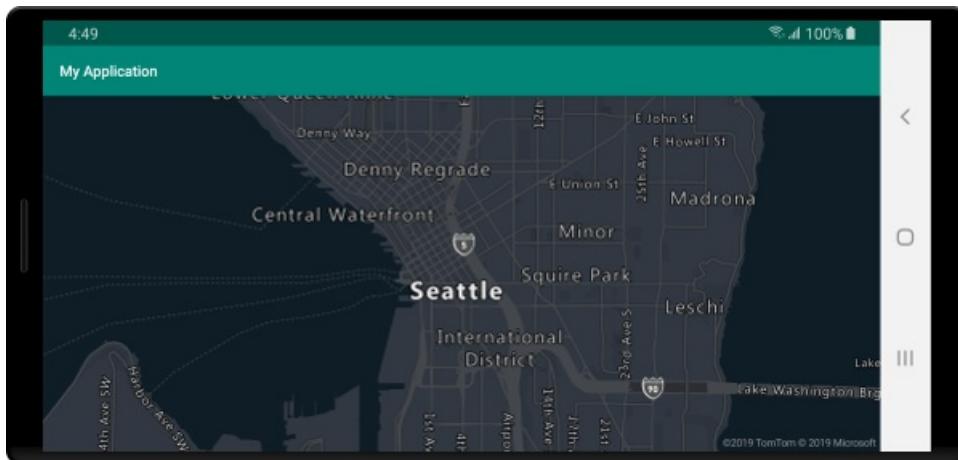
The procedure in this section requires an Azure Maps account in Gen 1 or Gen 2 pricing tier. For more information on pricing tiers, see [Choose the right pricing tier in Azure Maps](#).

Set map style in the layout

You can set a map style in the layout file for your activity class when adding the map control. The following code sets the center location, zoom level, and map style.

```
<com.microsoft.azure.maps.mapcontrol.MapControl  
    android:id="@+id/mapcontrol"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    app:mapcontrol_centerLat="47.602806"  
    app:mapcontrol_centerLng="-122.329330"  
    app:mapcontrol_zoom="12"  
    app:mapcontrol_style="grayscale_dark"  
/>
```

The following screenshot shows the above code displaying a road map with the grayscale dark style.



Set map style in code

The map style can be set in programmatically in code by using the `setStyle` method of the map. The following code sets the center location and zoom level using the `maps.setCamera` method and the map style to `SATELLITE_ROAD_LABELS`.

```
mapControl.onReady(map -> {

    //Set the camera of the map.
    map.setCamera(center(Point.fromLngLat(-122.33, 47.64)), zoom(14));

    //Set the style of the map.
    map.setStyle(style(MapStyle.SATELLITE_ROAD_LABELS));
});
```

```
mapControl!! .onReady { map: AzureMap ->
    //Set the camera of the map.
    map.setCamera(center(Point.fromLngLat(-122.33, 47.64)), zoom(14))

    //Set the style of the map.
    map.setStyle(style(MapStyle.SATELLITE_ROAD_LABELS))
}
```

The following screenshot shows the above code displaying a map with the satellite road labels style.



Setting the map camera

The map camera controls which part of the world is displayed in the map viewport. The camera can be in the layout our programmatically in code. When setting it in code, there are two main methods for setting the position of the map; using center and zoom, or passing in a bounding box. The following code shows how to set all optional camera options when using `center` and `zoom`.

```

//Set the camera of the map using center and zoom.
map.setCamera(
    center(Point.fromLngLat(-122.33, 47.64)),

    //The zoom level. Typically a value between 0 and 22.
    zoom(14),

    //The amount of tilt in degrees the map where 0 is looking straight down.
    pitch(45),

    //Direction the top of the map is pointing in degrees. 0 = North, 90 = East, 180 = South, 270 = West
    bearing(90),

    //The minimum zoom level the map will zoom-out to when animating from one location to another on the
    map.
    minZoom(10),

    //The maximum zoom level the map will zoom-in to when animating from one location to another on the map.
    maxZoom(14)
);

```

```

//Set the camera of the map using center and zoom.
map.setCamera(
    center(Point.fromLngLat(-122.33, 47.64)),

    //The zoom level. Typically a value between 0 and 22.
    zoom(14),

    //The amount of tilt in degrees the map where 0 is looking straight down.
    pitch(45),

    //Direction the top of the map is pointing in degrees. 0 = North, 90 = East, 180 = South, 270 = West
    bearing(90),

    //The minimum zoom level the map will zoom-out to when animating from one location to another on the
    map.
    minZoom(10),

    //The maximum zoom level the map will zoom-in to when animating from one location to another on the map.
    maxZoom(14)
)

```

Often it is desirable to focus the map over a set of data. A bounding box can be calculated from features using the `MapMath.fromData` method and can be passed into the `bounds` option of the map camera. When setting a map view based on a bounding box, it's often useful to specify a `padding` value to account for the pixel size of points being rendered as bubbles or symbols. The following code shows how to set all optional camera options when using a bounding box to set the position of the camera.

```

//Set the camera of the map using a bounding box.
map.setCamera(
    //The area to focus the map on.
    bounds(BoundingBox.fromLngLats(
        //West
        -122.4594,
        //South
        47.4333,
        //East
        -122.21866,
        //North
        47.75758
    )),
    //Amount of pixel buffer around the bounding box to provide extra space around the bounding box.
    padding(20),
    //The maximum zoom level the map will zoom-in to when animating from one location to another on the map.
    maxZoom(14)
);

```

```

//Set the camera of the map using a bounding box.
map.setCamera(
    //The area to focus the map on.
    bounds(BoundingBox.fromLngLats(
        //West
        -122.4594,
        //South
        47.4333,
        //East
        -122.21866,
        //North
        47.75758
    )),
    //Amount of pixel buffer around the bounding box to provide extra space around the bounding box.
    padding(20),
    //The maximum zoom level the map will zoom-in to when animating from one location to another on the map.
    maxZoom(14)
)

```

The aspect ratio of a bounding box may not be the same as the aspect ratio of the map, as such the map will often show the full bounding box area, but will often only be tight vertically or horizontally.

Next steps

See the following articles for more code samples to add to your maps:

[Add a symbol layer](#)

[Add a bubble layer](#)

Add controls to a map (Android SDK)

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to add UI controls to the map.

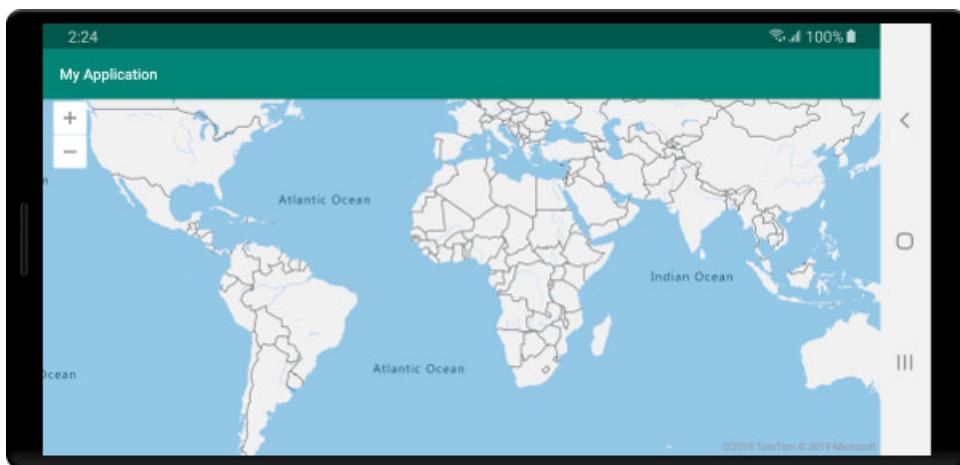
Add zoom control

A zoom control adds buttons for zooming the map in and out. The following code sample creates an instance of the `ZoomControl` class and adds it to a map.

```
//Construct a zoom control and add it to the map.  
map.controls.add(new ZoomControl());
```

```
//Construct a zoom control and add it to the map.  
map.controls.add(ZoomControl())
```

The screenshot below is of a zoom control loaded on a map.



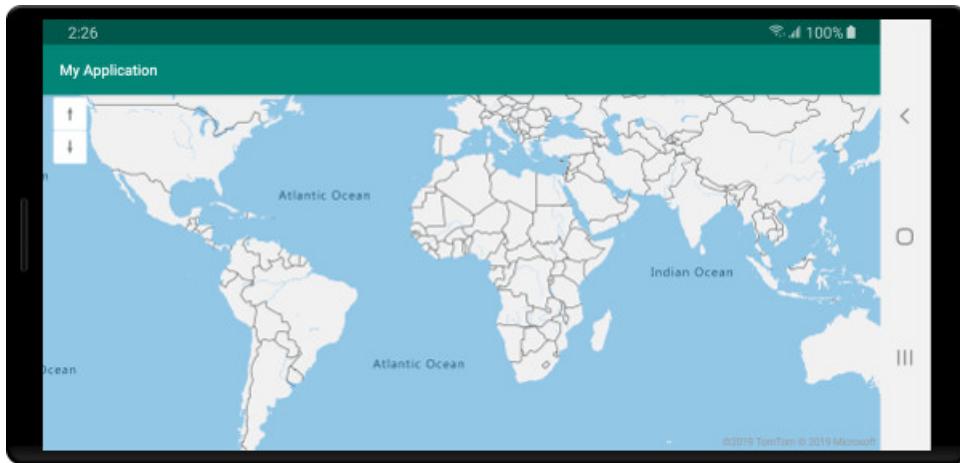
Add pitch control

A pitch control adds buttons for tilting the pitch to map relative to the horizon. The following code sample creates an instance of the `PitchControl` class and adds it to a map.

```
//Construct a pitch control and add it to the map.  
map.controls.add(new PitchControl());
```

```
//Construct a pitch control and add it to the map.  
map.controls.add(PitchControl())
```

The screenshot below is of a pitch control loaded on a map.



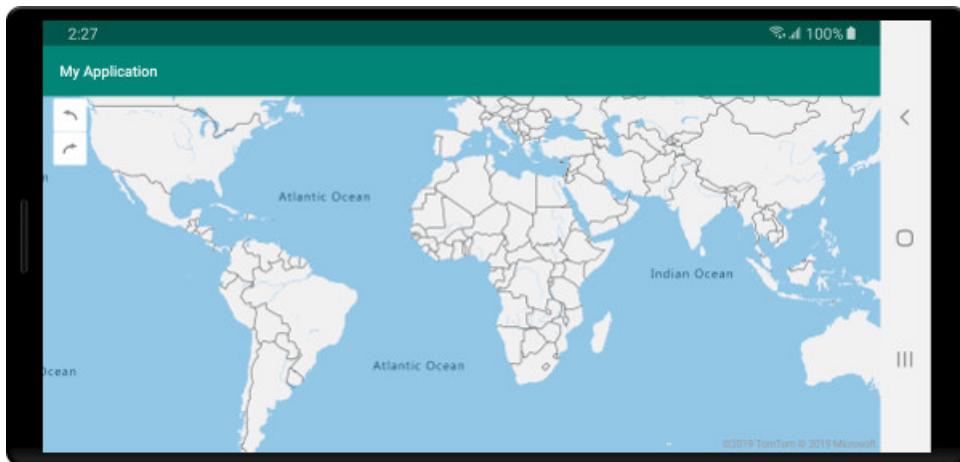
Add compass control

A compass control adds a button for rotating the map. The following code sample creates an instance of the `CompassControl` class and adds it to a map.

```
//Construct a compass control and add it to the map.  
map.controls.add(new CompassControl());
```

```
//Construct a compass control and add it to the map.  
map.controls.add(CompassControl())
```

The screenshot below is of a compass control loaded on a map.



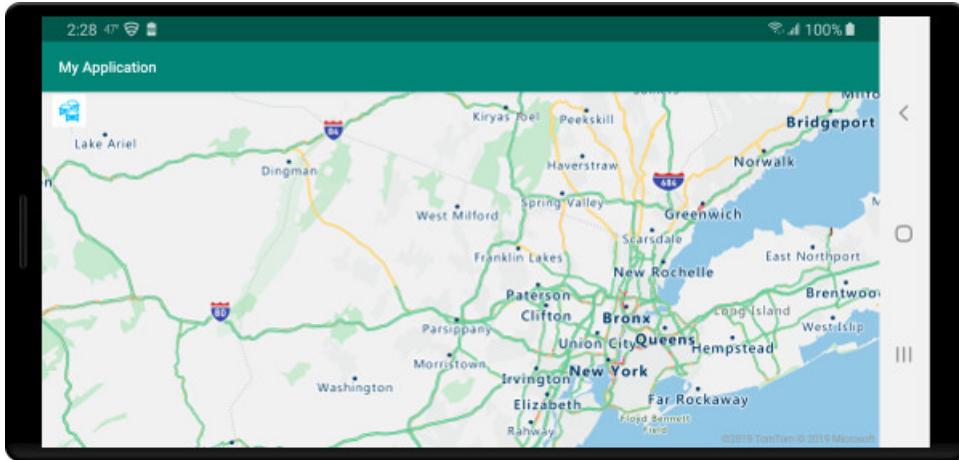
Add traffic control

A traffic control adds a button for toggling the visibility of traffic data on the map. The following code sample creates an instance of the `TrafficControl` class and adds it to a map.

```
//Construct a traffic control and add it to the map.  
map.controls.add(new TrafficControl());
```

```
//Construct a traffic control and add it to the map.  
map.controls.add(TrafficControl())
```

The screenshot below is of a traffic control loaded on a map.



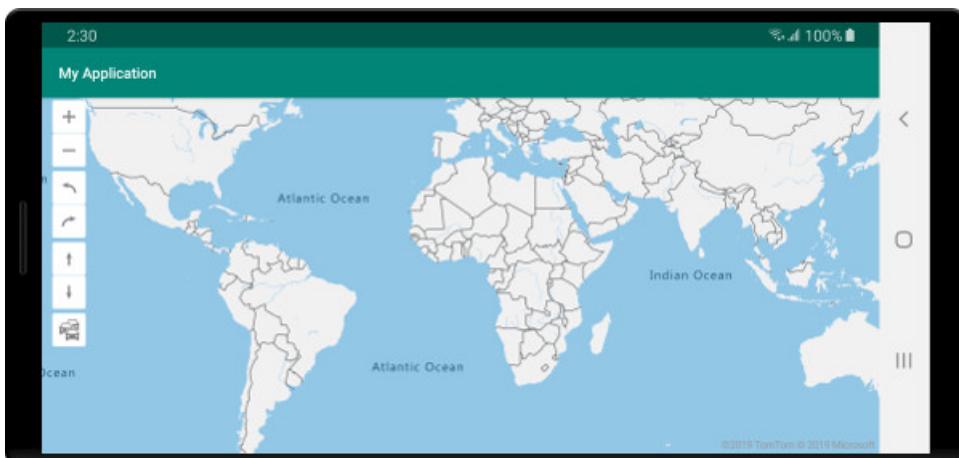
A Map with all controls

Multiple controls can be put into an array and added to the map all at once and positioned in the same area of the map to simplify development. The following adds the standard navigation controls to the map using this approach.

```
map.controls.add(  
    new Control[]{  
        new ZoomControl(),  
        new CompassControl(),  
        new PitchControl(),  
        new TrafficControl()  
    }  
);
```

```
map.controls.add(  
    arrayOf<Control>(  
        ZoomControl(),  
        CompassControl(),  
        PitchControl(),  
        TrafficControl()  
    )  
)
```

The screenshot below shows all controls loaded on a map. Note that the order they are added to the map, is the order they will appear.



Next steps

See the following articles for more code samples to add to your maps:

[Add a symbol layer](#)

[Add a bubble layer](#)

[Add a line layer](#)

[Add a polygon layer](#)

Create a data source (Android SDK)

6/1/2021 • 17 minutes to read • [Edit Online](#)

The Azure Maps Android SDK stores data in data sources. Using data sources optimizes the data operations for querying and rendering. Currently there are two types of data sources:

- **GeoJSON source:** Manages raw location data in GeoJSON format locally. Good for small to medium data sets (upwards of hundreds of thousands of shapes).
- **Vector tile source:** Loads data formatted as vector tiles for the current map view, based on the maps tiling system. Ideal for large to massive data sets (millions or billions of shapes).

GeoJSON data source

Azure Maps uses GeoJSON as one of its primary data models. GeoJSON is an open geospatial standard way for representing geospatial data in JSON format. GeoJSON classes available in the Azure Maps Android SDK to easily create, and serialize GeoJSON data. Load and store GeoJSON data in the `DataSource` class and render it using layers. The following code shows how GeoJSON objects can be created in Azure Maps.

```
/*
Raw GeoJSON feature

{
    "type": "Feature",
    "geometry": {
        "type": "Point",
        "coordinates": [-100, 45]
    },
    "properties": {
        "custom-property": "value"
    }
}

//Create a point feature.
Feature feature = Feature.fromGeometry(Point.fromLngLat(-100, 45));

//Add a property to the feature.
feature.addStringProperty("custom-property", "value");

//Add the feature to the data source.
source.add(feature);
```

```

/*
Raw GeoJSON feature

{
    "type": "Feature",
    "geometry": {
        "type": "Point",
        "coordinates": [-100, 45]
    },
    "properties": {
        "custom-property": "value"
    }
}

//Create a point feature.
val feature = Feature.fromGeometry(Point.fromLngLat(-100, 45))

//Add a property to the feature.
feature.addStringProperty("custom-property", "value")

//Add the feature to the data source.
source.add(feature)

```

Alternatively the properties can be loaded into a JsonObject first then passed into the feature when creating it, as shown below.

```

//Create a JsonObject to store properties for the feature.
JsonObject properties = new JsonObject();
properties.addProperty("custom-property", "value");

Feature feature = Feature.fromGeometry(Point.fromLngLat(-100, 45), properties);

```

```

//Create a JsonObject to store properties for the feature.
val properties = JsonObject()
properties.addProperty("custom-property", "value")

val feature = Feature.fromGeometry(Point.fromLngLat(-100, 45), properties)

```

Once you have a GeoJSON feature created, a data source can be added to the map through the `sources` property of the map. The following code shows how to create a `DataSource`, add it to the map, and add a feature to the data source.

```

//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Add GeoJSON feature to the data source.
source.add(feature);

```

The following code shows several ways to create a GeoJSON Feature, FeatureCollection, and geometries.

```
//GeoJSON Point Geometry
Point point = Point.fromLngLat(LONGITUDE, LATITUDE);

//GeoJSON Point Geometry
LineString linestring = LineString.fromLngLats(PointList);

//GeoJSON Polygon Geometry
Polygon polygon = Polygon.fromLngLats(listOfPointList);

Polygon polygonFromOuterInner = Polygon.fromOuterInner(outerLineStringObject,innerLineStringObject);

//GeoJSON MultiPoint Geometry
MultiPoint multiPoint = MultiPoint.fromLngLats(PointList);

//GeoJSON MultiLineString Geometry
MultiLineString multiLineStringFromLngLat = MultiLineString.fromLngLats(listOfPointList);

MultiLineString multiLineString = MultiLineString.fromLineString(singleLineString);

//GeoJSON MultiPolygon Geometry
MultiPolygon multiPolygon = MultiPolygon.fromLngLats(listOflistOfPointList);

MultiPolygon multiPolygonFromPolygon = MultiPolygon.fromPolygon(polygon);

MultiPolygon multiPolygonFromPolygons = MultiPolygon.fromPolygons(PolygonList);

//GeoJSON Feature
Feature pointFeature = Feature.fromGeometry(Point.fromLngLat(LONGITUDE, LATITUDE));

//GeoJSON FeatureCollection
FeatureCollection featureCollectionFromSingleFeature = FeatureCollection.fromFeature(pointFeature);

FeatureCollection featureCollection = FeatureCollection.fromFeatures(listOfFeatures);
```

```

//GeoJSON Point Geometry
val point = Point.fromLngLat(LONGITUDE, LATITUDE)

//GeoJSON Point Geometry
val linestring = LineString.fromLngLats(PointList)

//GeoJSON Polygon Geometry
val polygon = Polygon.fromLngLats(listOfPointList)

val polygonFromOuterInner = Polygon.fromOuterInner(outerLineStringObject, innerLineStringObject)

//GeoJSON MultiPoint Geometry
val multiPoint = MultiPoint.fromLngLats(PointList)

//GeoJSON MultiLineString Geometry
val multiLineStringFromLngLat = MultiLineString.fromLngLats(listOfPointList)

val multiLineString = MultiLineString.fromLineString(singleLineString)

//GeoJSON MultiPolygon Geometry
val multiPolygon = MultiPolygon.fromLngLats(listOflistOfPointList)

val multiPolygonFromPolygon = MultiPolygon.fromPolygon(polygon)

val multiPolygonFromPolygons = MultiPolygon.fromPolygons(PolygonList)

//GeoJSON Feature
val pointFeature = Feature.fromGeometry(Point.fromLngLat(LONGITUDE, LATITUDE))

//GeoJSON FeatureCollection
val featureCollectionFromSingleFeature = FeatureCollection.fromFeature(pointFeature)

val featureCollection = FeatureCollection.fromFeatures(listOfFeatures)

```

Serialize and deserialize GeoJSON

The feature collection, feature, and geometry classes all have `fromJson()` and `toJson()` static methods, which help with serialization. The formatted valid JSON String passed through the `fromJson()` method will create the geometry object. This `fromJson()` method also means you can use Gson or other serialization/deserialization strategies. The following code shows how to take a stringified GeoJSON feature and deserialize it into the Feature class, then serialize it back into a GeoJSON string.

```

//Take a stringified GeoJSON object.
String GeoJSON_STRING = "{"  
    + "    \"type\": \"Feature\","  
    + "    \"geometry\": {"  
    + "        \"type\": \"Point\","  
    + "        \"coordinates\": [-100, 45]  
    + "    },"  
    + "    \"properties\": {"  
    + "        \"custom-property\": \"value\""  
    + "    },"  
    + "}";  
  
//Deserialize the JSON string into a feature.  
Feature feature = Feature.fromJson(GeoJSON_STRING);  
  
//Serialize a feature collection to a string.  
String featureString = feature.toJson();

```

```

//Take a stringified GeoJSON object.
val GeoJSON_STRING = ("{
    + "      \\"type\\": \\"Feature\",
    + "      \\"geometry\\": {
        + "          \\"type\\": \\"Point\"
        + "          \\"coordinates\\": [-100, 45]
    + "      },
    + "      \\"properties\\": {
        + "          \\"custom-property\\": \\"value\"
    + "      }
}"))
}

//Deserialize the JSON string into a feature.
val feature = Feature.fromJson(GeoJSON_STRING)

//Serialize a feature collection to a string.
val featureString = feature.toJson()

```

Import GeoJSON data from web or assets folder

Most GeoJSON files contain a FeatureCollection. Read GeoJSON files as strings and used the `FeatureCollection.fromJson` method to deserialize it.

The following code is a reusable class for importing data from the web or local assets folder as a string and returning it to the UI thread via a callback function.

```

import android.content.Context;
import android.os.Handler;
import android.os.Looper;
import android.webkit.URLUtil;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import javax.net.ssl.HttpsURLConnection;

public class Utils {

    interface SimpleCallback {
        void notify(String result);
    }

    /**
     * Imports data from a web url or asset file name and returns it to a callback.
     * @param urlOrFileName A web url or asset file name that points to data to load.
     * @param context The context of the app.
     * @param callback The callback function to return the data to.
     */
    public static void importData(String urlOrFileName, Context context, SimpleCallback callback){
        importData(urlOrFileName, context, callback, null);
    }

    /**
     * Imports data from a web url or asset file name and returns it to a callback.
     * @param urlOrFileName A web url or asset file name that points to data to load.
     * @param context The context of the app.
     * @param callback The callback function to return the data to.
     * @param error A callback function to return errors to.
     */
    public static void importData(String urlOrFileName, Context context, SimpleCallback callback,
        SimpleCallback error){
}

```

```

import java.util.concurrent.Executor;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import android.os.Handler;
import android.os.Looper;

public class DataImportUtil {

    /**
     * Imports data from a URL or file name as a string.
     * @param urlOrFileName URL or file name to import.
     * @param callback The callback to receive the imported data.
     */
    public static void importData(String urlOrFileName, final ImportCallback callback) {
        if(urlOrFileName != null && callback != null) {
            ExecutorService executor = Executors.newSingleThreadExecutor();
            Handler handler = new Handler(Looper.getMainLooper());

            executor.execute(() -> {
                String data = null;

                try {

                    if(URLUtil.isNetworkUrl(urlOrFileName)){
                        data = importFromWeb(urlOrFileName);
                    } else {
                        //Assume file is in assets folder.
                        data = importFromAssets(context, urlOrFileName);
                    }

                    final String result = data;

                    handler.post(() -> {
                        //Ensure the resulting data string is not null or empty.
                        if (result != null && !result.isEmpty()) {
                            callback.notify(result);
                        } else {
                            error.notify("No data imported.");
                        }
                    });
                } catch(Exception e) {
                    if(error != null){
                        error.notify(e.getMessage());
                    }
                }
            });
        }
    }

    /**
     * Imports data from an assets file as a string.
     * @param context The context of the app.
     * @param fileName The asset file name.
     * @return
     * @throws IOException
     */
    private static String importFromAssets(Context context, String fileName) throws IOException {
        InputStream stream = null;

        try {
            stream = context.getAssets().open(fileName);

            if(stream != null) {
                return readStreamAsString(stream);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // Close Stream and disconnect HTTPS connection.
            if (stream != null) {
                stream.close();
            }
        }

        return null;
    }

    /**
     * Imports data from the web as a string.
     * @param url URL to the data.
     * @return
     * @throws IOException
     */

```

```

/*
private static String importFromWeb(String url) throws IOException {
    InputStream stream = null;
    HttpsURLConnection connection = null;
    String result = null;

    try {
        connection = (HttpsURLConnection) new URL(url).openConnection();

        //For this use case, set HTTP method to GET.
        connection.setRequestMethod("GET");

        //Open communications link (network traffic occurs here).
        connection.connect();

        int responseCode = connection.getResponseCode();
        if (responseCode != HttpsURLConnection.HTTP_OK) {
            throw new IOException("HTTP error code: " + responseCode);
        }

        //Retrieve the response body as an InputStream.
        stream = connection.getInputStream();

        if (stream != null) {
            return readStreamAsString(stream);
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // Close Stream and disconnect HTTPS connection.
        if (stream != null) {
            stream.close();
        }
        if (connection != null) {
            connection.disconnect();
        }
    }

    return result;
}

/**
 * Reads an input stream as a string.
 * @param stream Stream to convert.
 * @return
 * @throws IOException
 */
private static String readStreamAsString(InputStream stream) throws IOException {
    //Convert the contents of an InputStream to a String.
    BufferedReader in = new BufferedReader(new InputStreamReader(stream, "UTF-8"));

    String inputLine;
    StringBuffer response = new StringBuffer();

    while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
    }

    in.close();

    return response.toString();
}
}

```

```

import android.content.Context
import android.os.Handler
import android.os.Looper
import android.webkit.URLUtil
import java.net.URL
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors

class Utils {
    companion object {

        /**
         * Imports data from a web url or asset file name and returns it to a callback.
         * @param urlOrFileName A web url or asset file name that points to data to load.
         * @param context The context of the app.
         * @param callback The callback function to return the data to.
         */
        fun importData(urlOrFileName: String?, context: Context, callback: (String?) -> Unit) {
            importData(urlOrFileName, context, callback, null)
        }

        /**
         * Imports data from a web url or asset file name and returns it to a callback.
         * @param urlOrFileName A web url or asset file name that points to data to load.
         * @param context The context of the app.
         * @param callback The callback function to return the data to.
         * @param error A callback function to return errors to.
         */
        public fun importData(urlOrFileName: String?, context: Context, callback: (String?) -> Unit, error: ((String?) -> Unit)?) {
            if (urlOrFileName != null && callback != null) {
                val executor: ExecutorService = Executors.newSingleThreadExecutor()
                val handler = Handler(Looper.getMainLooper())
                executor.execute {
                    var data: String? = null

                    try {
                        data = if (URLUtil.isNetworkUrl(urlOrFileName)) {
                            URL(urlOrFileName).readText()
                        } else { //Assume file is in assets folder.
                            context.assets.open(urlOrFileName).bufferedReader().use{
                                it.readText()
                            }
                        }

                        handler.post {
                            //Ensure the resulting data string is not null or empty.
                            if (data != null && !data.isEmpty()) {
                                callback(data)
                            } else {
                                error!("No data imported.")
                            }
                        }
                    } catch (e: Exception) {
                        error!(e.message)
                    }
                }
            }
        }
    }
}

```

The code below shows how to use this utility to import GeoJSON data as a string and return it to the UI thread via a callback. In the callback, the string data can be serialized into a GeoJSON Feature collection and added to the data source. Optionally, update the maps camera to focus in on the data.

```

//Create a data source and add it to the map.
DataSource dataSource = new DataSource();
map.sources.add(dataSource);

//Import the geojson data and add it to the data source.
Utils.importData("URL_or_FilePath_to_GeoJSON_data",
    this,
    (String result) -> {
        //Parse the data as a GeoJSON Feature Collection.
        FeatureCollection fc = FeatureCollection.fromJson(result);

        //Add the feature collection to the data source.
        dataSource.add(fc);

        //Optionally, update the maps camera to focus in on the data.

        //Calculate the bounding box of all the data in the Feature Collection.
        BoundingBox bbox = MapMath.fromData(fc);

        //Update the maps camera so it is focused on the data.
        map.setCamera(
            bounds(bbox),
            padding(20));
    });

```

```

//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Import the GeoJSON data and add it to the data source.
Utils.importData("SamplePoiDataSet.json", this) {
    result: String? ->
        //Parse the data as a GeoJSON Feature Collection.
        val fc = FeatureCollection.fromJson(result!!)

        //Add the feature collection to the data source.
        source.add(fc)

        //Optionally, update the maps camera to focus in on the data.

        //Calculate the bounding box of all the data in the Feature Collection.
        val bbox = MapMath.fromData(fc);

        //Update the maps camera so it is focused on the data.
        map.setCamera(
            bounds(bbox),

            //Padding added to account for pixel size of rendered points.
            padding(20)
        )
}

```

Vector tile source

A vector tile source describes how to access a vector tile layer. Use the `VectorTileSource` class to instantiate a vector tile source. Vector tile layers are similar to tile layers, but they aren't the same. A tile layer is a raster image. Vector tile layers are a compressed file, in **PBF** format. This compressed file contains vector map data, and one or more layers. The file can be rendered and styled on the client, based on the style of each layer. The data in a vector tile contain geographic features in the form of points, lines, and polygons. There are several advantages of using vector tile layers instead of raster tile layers:

- A file size of a vector tile is typically much smaller than an equivalent raster tile. As such, less bandwidth is

used. It means lower latency, a faster map, and a better user experience.

- Since vector tiles are rendered on the client, they adapt to the resolution of the device they're being displayed on. As a result, the rendered maps appear more well defined, with crystal clear labels.
- Changing the style of the data in the vector maps doesn't require downloading the data again, since the new style can be applied on the client. In contrast, changing the style of a raster tile layer typically requires loading tiles from the server then applying the new style.
- Since the data is delivered in vector form, there's less server-side processing required to prepare the data. As a result, the newer data can be made available faster.

Azure Maps adheres to the [Mapbox Vector Tile Specification](#), an open standard. Azure Maps provides the following vector tiles services as part of the platform:

- Road tiles [documentation](#) | [data format details](#)
- Traffic incidents [documentation](#) | [data format details](#)
- Traffic flow [documentation](#) | [data format details](#)
- Azure Maps Creator also allows custom vector tiles to be created and accessed through the [Render V2-Get Map Tile API](#)

TIP

When using vector or raster image tiles from the Azure Maps render service with the web SDK, you can replace `atlas.microsoft.com` with the placeholder `azmapsdomain.invalid`. This placeholder will be replaced with the same domain used by the map and will automatically append the same authentication details as well. This greatly simplifies authentication with the render service when using Azure Active Directory authentication.

To display data from a vector tile source on the map, connect the source to one of the data rendering layers. All layers that use a vector source must specify a `sourceLayer` value in the options. The following code loads the Azure Maps traffic flow vector tile service as a vector tile source, then displays it on a map using a line layer. This vector tile source has a single set of data in the source layer called "Traffic flow". The line data in this data set has a property called `traffic_level` that is used in this code to select the color and scale the size of lines.

```
//Formatted URL to the traffic flow vector tiles, with the maps subscription key appended to it.
String trafficFlowUrl = "https://azmapsdomain.invalid/traffic/flow/tile/pbf?api-
version=1.0&style=relative&zoom={z}&x={x}&y={y}";

//Create a vector tile source and add it to the map.
VectorTileSource source = new VectorTileSource(
    tiles(new String[] { trafficFlowUrl }),
    maxSourceZoom(22)
);
map.sources.add(source);

//Create a layer for traffic flow lines.
LineLayer layer = new LineLayer(source,
    //The name of the data layer within the data source to pass into this rendering layer.
    sourceLayer("Traffic flow"),

    //Color the roads based on the traffic_level property.
    strokeColor(
        interpolate(
            linear(),
            get("traffic_level"),
            stop(0, color(Color.RED)),
            stop(0.33, color(Color.YELLOW)),
            stop(0.66, color(Color.GREEN))
        )
    ),
    //Scale the width of roads based on the traffic_level property.
    strokeWidth(
        interpolate(
            linear(),
            get("traffic_level"),
            stop(0, 6),
            stop(1,1)
        )
    )
);
map.layers.add(layer, "labels");
```

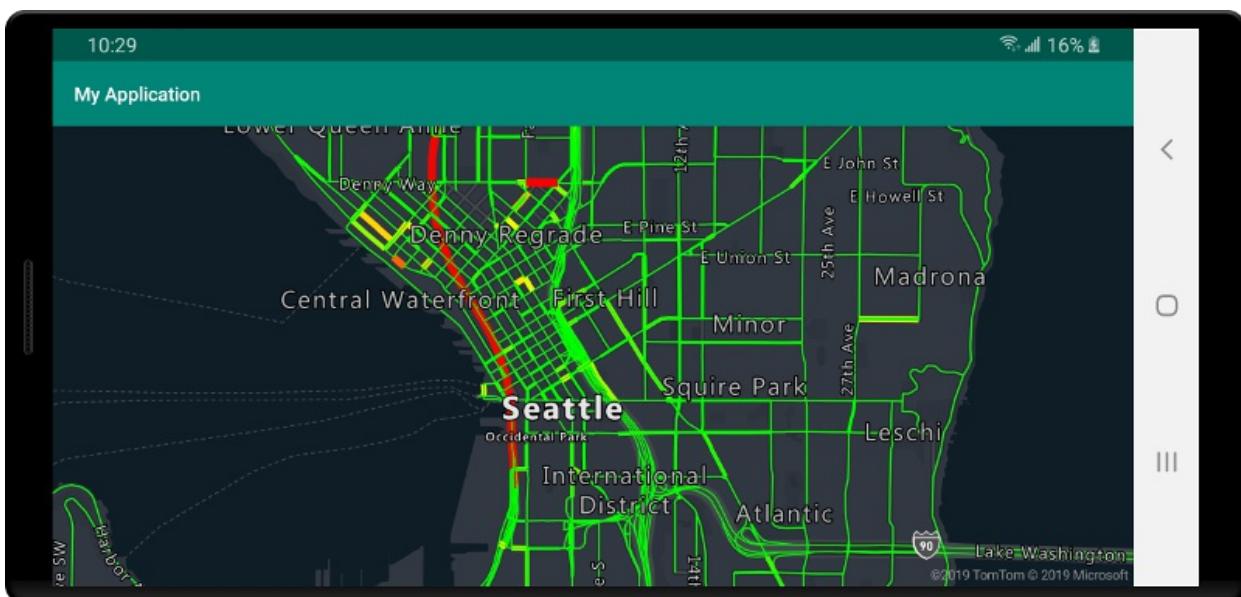
```

//Formatted URL to the traffic flow vector tiles, with the maps subscription key appended to it.
val trafficFlowUrl = "https://azmapsdomain.invalid/traffic/flow/tile/pbf?api-
version=1.0&style=relative&zoom={z}&x={x}&y={y}"

//Create a vector tile source and add it to the map.
val source = VectorTileSource(
    tiles(arrayOf(trafficFlowUrl)),
    maxSourceZoom(22)
)
map.sources.add(source)

//Create a layer for traffic flow lines.
val layer = LineLayer(
    source, //The name of the data layer within the data source to pass into this rendering layer.
    sourceLayer("Traffic flow"), //Color the roads based on the traffic_level property.
    strokeColor(
        interpolate(
            linear(),
            get("traffic_level"),
            stop(0, color(Color.RED)),
            stop(0.33, color(Color.YELLOW)),
            stop(0.66, color(Color.GREEN))
        )
    ), //Scale the width of roads based on the traffic_level property.
    strokeWidth(
        interpolate(
            linear(),
            get("traffic_level"),
            stop(0, 6),
            stop(1, 1)
        )
    )
)
map.layers.add(layer, "labels")

```



Connecting a data source to a layer

Data is rendered on the map using rendering layers. A single data source can be referenced by one or more rendering layers. The following rendering layers require a data source:

- **Bubble layer** - renders point data as scaled circles on the map.
- **Symbol layer** - renders point data as icons or text.

- [Heat map layer](#) - renders point data as a density heat map.
- [Line layer](#) - render a line and or render the outline of polygons.
- [Polygon layer](#) - fills the area of a polygon with a solid color or image pattern.

The following code shows how to create a data source, add it to the map, and connect it to a bubble layer. And then, import GeoJSON point data from a remote location into the data source.

```
//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Create a layer that defines how to render points in the data source and add it to the map.
BubbleLayer layer = new BubbleLayer(source);
map.layers.add(layer);

//Import the geojson data and add it to the data source.
Utils.importData("URL_or_FilePath_to_GeoJSON_data",
    this,
    (String result) -> {
        //Parse the data as a GeoJSON Feature Collection.
        FeatureCollection fc = FeatureCollection.fromJson(result);

        //Add the feature collection to the data source.
        dataSource.add(fc);

        //Optionally, update the maps camera to focus in on the data.

        //Calculate the bounding box of all the data in the Feature Collection.
        BoundingBox bbox = MapMath.fromData(fc);

        //Update the maps camera so it is focused on the data.
        map.setCamera(
            bounds(bbox),
            padding(20));
    });
}
```

```
//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

//Create a layer that defines how to render points in the data source and add it to the map.
val layer = BubbleLayer(source)
map.layers.add(layer)

//Import the geojson data and add it to the data source.
Utils.importData("URL_or_FilePath_to_GeoJSON_data", this) {
    result: String? ->
        //Parse the data as a GeoJSON Feature Collection.
        val fc = FeatureCollection.fromJson(result!!)

        //Add the feature collection to the data source.
        dataSource.add(fc)

        //Optionally, update the maps camera to focus in on the data.
        //Calculate the bounding box of all the data in the Feature Collection.
        val bbox = MapMath.fromData(fc)

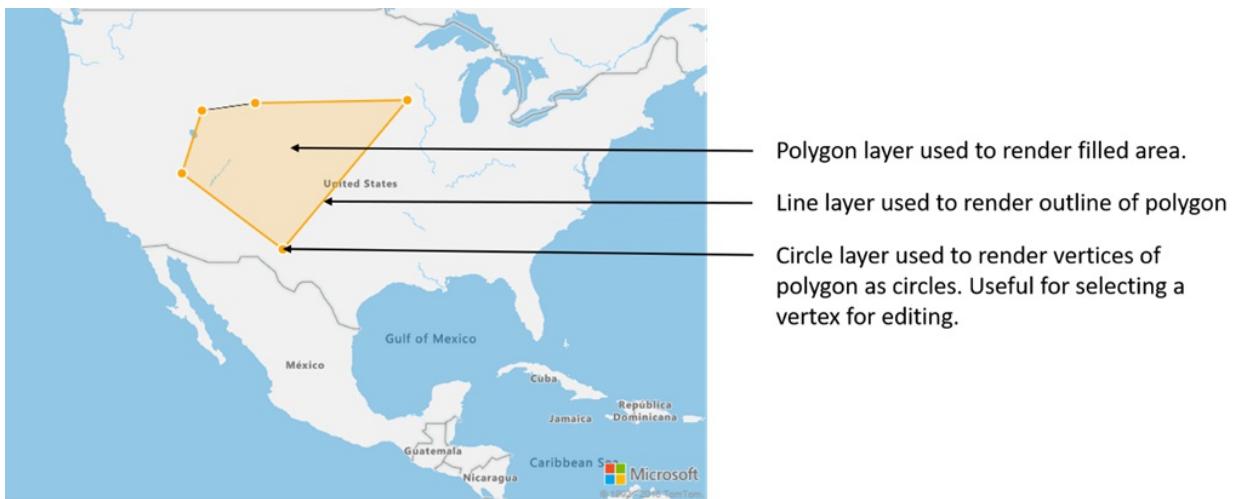
        //Update the maps camera so it is focused on the data.
        map.setCamera(
            bounds(bbox),
            padding(20)
        )
}
```

There are additional rendering layers that don't connect to these data sources, but they directly load the data for rendering.

- [Tile layer](#) - superimposes a raster tile layer on top of the map.

One data source with multiple layers

Multiple layers can be connected to a single data source. There are many different scenarios in which this option is useful. For example, consider the scenario in which a user draws a polygon. We should render and fill the polygon area as the user adds points to the map. Adding a styled line to outline the polygon makes it easier see the edges of the polygon, as the user draws. To conveniently edit an individual position in the polygon, we may add a handle, like a pin or a marker, above each position.



In most mapping platforms, you would need a polygon object, a line object, and a pin for each position in the polygon. As the polygon is modified, you would need to manually update the line and pins, which can quickly become complex.

With Azure Maps, all you need is a single polygon in a data source as shown in the code below.

```

//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Create a polygon and add it to the data source.
source.add(Polygon.fromLngLats(/* List of points */));

//Create a polygon layer to render the filled in area of the polygon.
PolygonLayer polygonLayer = new PolygonLayer(source,
    fillColor("rgba(255,165,0,0.2)")
);

//Create a line layer for greater control of rendering the outline of the polygon.
LineLayer lineLayer = new LineLayer(source,
    strokeColor("orange"),
    strokeWidth(2f)
);

//Create a bubble layer to render the vertices of the polygon as scaled circles.
BubbleLayer bubbleLayer = new BubbleLayer(source,
    bubbleColor("orange"),
    bubbleRadius(5f),
    bubbleStrokeColor("white"),
    bubbleStrokeWidth(2f)
);

//Add all layers to the map.
map.layers.add(new Layer[] { polygonLayer, lineLayer, bubbleLayer });

```

```

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

//Create a polygon and add it to the data source.
source.add(Polygon.fromLngLats())

//Create a polygon layer to render the filled in area of the polygon.
val polygonLayer = PolygonLayer(
    source,
    fillColor("rgba(255,165,0,0.2)")
)

//Create a line layer for greater control of rendering the outline of the polygon.
val lineLayer = LineLayer(
    source,
    strokeColor("orange"),
    strokeWidth(2f)
)

//Create a bubble layer to render the vertices of the polygon as scaled circles.
val bubbleLayer = BubbleLayer(
    source,
    bubbleColor("orange"),
    bubbleRadius(5f),
    bubbleStrokeColor("white"),
    bubbleStrokeWidth(2f)
)

//Add all layers to the map.
map.layers.add(arrayOf<Layer>(polygonLayer, lineLayer, bubbleLayer))

```

TIP

When adding layers to the map using the `map.layers.add` method, the ID or instance of an existing layer can be passed in as a second parameter. This would tell that map to insert the new layer being added below the existing layer. In addition to passing in a layer ID this method also supports the following values.

- `"labels"` - Inserts the new layer below the map label layers.
- `"transit"` - Inserts the new layer below the map road and transit layers.

Next steps

See the following articles for more code samples to add to your maps:

[Use data-driven style expressions](#)

[Add a symbol layer](#)

[Add a bubble layer](#)

[Add a line layer](#)

[Add a polygon layer](#)

[Add a heat map](#)

[Web SDK Code samples](#)

Add a symbol layer (Android SDK)

3/5/2021 • 6 minutes to read • [Edit Online](#)

This article shows you how to render point data from a data source as a symbol layer on a map using the Azure Maps Android SDK. Symbol layers render points as an image and text on the map.

TIP

Symbol layers by default will render the coordinates of all geometries in a data source. To limit the layer so that it only renders point geometry features, set the `filter` option of the layer to `eq(geometryType(), "Point")`. If you want to include MultiPoint features as well, set the `filter` option of the layer to `any(eq(geometryType(), "Point"), eq(geometryType(), "MultiPoint"))`.

Prerequisites

Be sure to complete the steps in the [Quickstart: Create an Android app](#) document. Code blocks in this article can be inserted into the maps `onReady` event handler.

Add a symbol layer

Before you can add a symbol layer to the map, you need to take a couple of steps. First, create a data source, and add it to the map. Create a symbol layer. Then, pass in the data source to the symbol layer, to retrieve the data from the data source. Finally, add data into the data source, so that there's something to be rendered.

The code below demonstrates what should be added to the map after it has loaded. This sample renders a single point on the map using a symbol layer.

```
//Create a data source and add it to the map.  
DataSource source = new DataSource();  
map.sources.add(source);  
  
//Create a point and add it to the data source.  
source.add(Point.fromLngLat(0, 0));  
  
//Create a symbol layer to render icons and/or text at points on the map.  
SymbolLayer layer = new SymbolLayer(source);  
  
//Add the layer to the map.  
map.layers.add(layer);
```

```
//Create a data source and add it to the map.  
val source = DataSource()  
map.sources.add(source)  
  
//Create a point and add it to the data source.  
source.add(Point.fromLngLat(0, 0))  
  
//Create a symbol layer to render icons and/or text at points on the map.  
val layer = SymbolLayer(source)  
  
//Add the layer to the map.  
map.layers.add(layer)
```

There are three different types of point data that can be added to the map:

- GeoJSON Point geometry - This object only contains a coordinate of a point and nothing else. The `Point.fromLngLat` static method can be used to easily create these objects.
- GeoJSON MultiPoint geometry - This object contains the coordinates of multiple points and nothing else. Pass an array of points into the `MultiPoint` class to create these objects.
- GeoJSON Feature - This object consists of any GeoJSON geometry and a set of properties that contain metadata associated to the geometry.

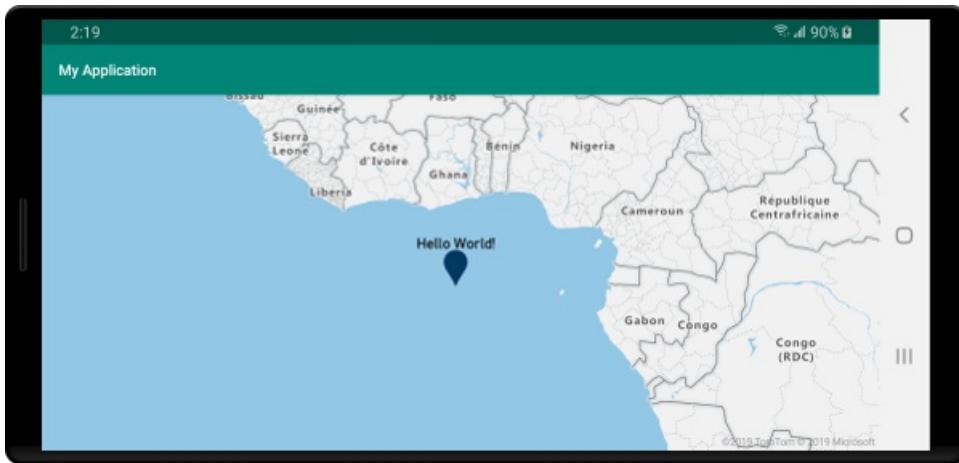
For more information, see the [Create a data source](#) document on creating and adding data to the map.

The following code sample creates a GeoJSON Point geometry and passes it into the GeoJSON Feature and has a `title` value added to its properties. The `title` property is displayed as text above the symbol icon on the map.

```
//Create a data source and add it to the map.  
DataSource source = new DataSource();  
map.sources.add(source);  
  
//Create a point feature.  
Feature feature = Feature.fromGeometry(Point.fromLngLat(0, 0));  
  
//Add a property to the feature.  
feature.addStringProperty("title", "Hello World!");  
  
//Add the feature to the data source.  
source.add(feature);  
  
//Create a symbol layer to render icons and/or text at points on the map.  
SymbolLayer layer = new SymbolLayer(source,  
    //Get the title property of the feature and display it on the map.  
    textField(get("title"))  
);  
  
//Add the layer to the map.  
map.layers.add(layer);
```

```
//Create a data source and add it to the map.  
val source = DataSource()  
map.sources.add(source)  
  
//Create a point feature.  
val feature = Feature.fromGeometry(Point.fromLngLat(0, 0))  
  
//Add a property to the feature.  
feature.addStringProperty("title", "Hello World!")  
  
//Add the feature to the data source.  
source.add(feature)  
  
//Create a symbol layer to render icons and/or text at points on the map.  
val layer = SymbolLayer(  
    source, //Get the title property of the feature and display it on the map.  
    textField(get("title"))  
)  
  
//Add the layer to the map.  
map.layers.add(layer)
```

The following screenshot shows the above code rendering a point feature using an icon and text label with a symbol layer.



TIP

By default, symbol layers optimize the rendering of symbols by hiding symbols that overlap. As you zoom in, the hidden symbols become visible. To disable this feature and render all symbols at all times, set the `iconAllowOverlap` and `textAllowOverlap` options to `true`.

Add a custom icon to a symbol layer

Symbol layers are rendered using WebGL. As such all resources, such as icon images, must be loaded into the WebGL context. This sample shows how to add a custom icon to the map resources. This icon is then used to render point data with a custom symbol on the map. The `textField` property of the symbol layer requires an expression to be specified. In this case, we want to render the temperature property. Since temperature is a number, it needs to be converted to a string. Additionally we want to append "°F" to it. An expression can be used to do this concatenation; `concat(Expression.toString(get("temperature")), literal("°F"))`.

```
//Load a custom icon image into the image sprite of the map.  
map.images.add("my-custom-icon", R.drawable.showers);  
  
//Create a data source and add it to the map.  
DataSource source = new DataSource();  
map.sources.add(source);  
  
//Create a point feature.  
Feature feature = Feature.fromGeometry(Point.fromLngLat(-73.985708, 40.75773));  
  
//Add a property to the feature.  
feature.addNumberProperty("temperature", 64);  
  
//Add the feature to the data source.  
source.add(feature);  
  
//Create a symbol layer to render icons and/or text at points on the map.  
SymbolLayer layer = new SymbolLayer(source,  
    iconImage("my-custom-icon"),  
    iconSize(0.5f),  
  
    //Get the title property of the feature and display it on the map.  
    textField(concat(Expression.toString(get("temperature")), literal("°F"))),  
    textOffset(new Float[]{0f, -1.5f})  
);
```

```

//Load a custom icon image into the image sprite of the map.
map.images.add("my-custom-icon", R.drawable.showers)

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

//Create a point feature.
val feature = Feature.fromGeometry(Point.fromLngLat(-73.985708, 40.75773))

//Add a property to the feature.
feature.addNumberProperty("temperature", 64)

//Add the feature to the data source.
source.add(feature)

//Create a symbol layer to render icons and/or text at points on the map.
val layer = SymbolLayer(
    source,
    iconImage("my-custom-icon"),
    iconSize(0.5f), //Get the title property of the feature and display it on the map.
    textField(concat(Expression.toString(get("temperature")), literal("°F"))),
    textOffset(arrayOf(0f, -1.5f))
)

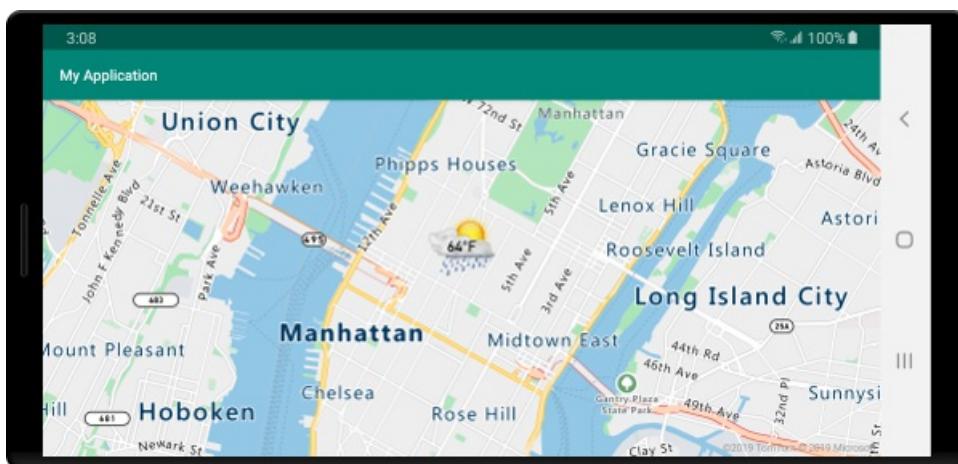
```

For this sample, the following image was loaded into the drawable folder of the app.



showers.png

The following screenshot shows the above code rendering a point feature using a custom icon and formatted text label with a symbol layer.



TIP

When you want to render only text with a symbol layer, you can hide the icon by setting the `iconImage` property of the icon options to `"none"`.

Modify symbol colors

The Azure Maps Android SDK comes with a set of predefined color variations of the default marker icon. For example, `marker-red` can be passed into the `iconImage` option of a symbol layer to render a red version of the marker icon in that layer.

```
SymbolLayer layer = new SymbolLayer(source,
    iconImage("marker-red")
);
```

```
val layer = SymbolLayer(source,
    iconImage("marker-red")
)
```

The table below lists all of the built-in icon image names available. All of these markers pull its colors from color resources that you can override. In addition to overriding the main fill color of this marker. However, note that overriding the color of one of these markers would apply to all layers that use that icon image.

ICON IMAGE NAME	COLOR RESOURCE NAME
<code>marker-default</code>	<code>mapcontrol_marker_default</code>
<code>marker-black</code>	<code>mapcontrol_marker_black</code>
<code>marker-blue</code>	<code>mapcontrol_marker_blue</code>
<code>marker-darkblue</code>	<code>mapcontrol_marker_darkblue</code>
<code>marker-red</code>	<code>mapcontrol_marker_red</code>
<code>marker-yellow</code>	<code>mapcontrol_marker_yellow</code>

You can also override the border color of all markers using the `mapcontrol_marker_border` color resource name. The colors of these markers can be overridden by adding a color with the same name in the `colors.xml` file of your app. For example, the following `colors.xml` file would make the default marker color bright green.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="mapcontrol_marker_default">#00FF00</color>
</resources>
```

The following is a modified version of the default marker vector XML that you can modify to create additional custom versions of the default marker. The modified version can be added to the `drawable` folder of your app and added to the maps image sprite using `map.images.add`, then used with a symbol layer.

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24.5dp"
    android:height="36.5dp"
    android:viewportWidth="24.5"
    android:viewportHeight="36.5">
    <path
        android:pathData="M12.25,0.25a12.2543,12.2543 0,0 0,-12 12.4937c0,6.4436 6.4879,12.1093
11.059,22.5641 0.5493,1.2563 1.3327,1.2563 1.882,0C17.7621,24.8529 24.25,19.1857
24.25,12.7437A12.2543,12.2543 0,0 0,12.25 0.25Z"
        android:strokeWidth="0.5"
        android:fillColor="@color/mapcontrol_marker_default"
        android:strokeColor="@color/mapcontrol_marker_border"/>
</vector>
```

Next steps

See the following articles for more code samples to add to your maps:

[Create a data source](#)

[Add a bubble layer](#)

[Use data-driven style expressions](#)

[Display feature information](#)

Add a bubble layer to a map (Android SDK)

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to render point data from a data source as a bubble layer on a map. Bubble layers render points as circles on the map with a fixed pixel radius.

TIP

Bubble layers by default will render the coordinates of all geometries in a data source. To limit the layer so that it only renders point geometry features, set the `filter` option of the layer to `eq(geometryType(), "Point")`. If you want to include MultiPoint features as well, set the `filter` option of the layer to `any(eq(geometryType(), "Point"), eq(geometryType(), "MultiPoint"))`.

Prerequisites

Be sure to complete the steps in the [Quickstart: Create an Android app](#) document. Code blocks in this article can be inserted into the maps `onReady` event handler.

Add a bubble layer

The following code loads an array of points into a data source. Then, it connects the data points are to a bubble layer. The bubble layer renders the radius of each bubble with five pixels and a fill color of white. And, a stroke color of blue, and a stroke width of six pixels.

```
//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Create point locations.
Point[] points = new Point[] {
    Point.fromLatLng(-73.985708, 40.75773),
    Point.fromLatLng(-73.985600, 40.76542),
    Point.fromLatLng(-73.985550, 40.77900),
    Point.fromLatLng(-73.975550, 40.74859),
    Point.fromLatLng(-73.968900, 40.78859)
};

//Add multiple points to the data source.
source.add(points);

//Create a bubble layer to render the filled in area of the circle, and add it to the map.
BubbleLayer layer = new BubbleLayer(source,
    bubbleRadius(5f),
    bubbleColor("white"),
    bubbleStrokeColor("#4288f7"),
    bubbleStrokeWidth(6f)
);

map.layers.add(layer);
```

```

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

//Create point locations.
val points: Array<Point> = arrayOf<Point>(
    Point.fromLngLat(-73.985708, 40.75773),
    Point.fromLngLat(-73.985600, 40.76542),
    Point.fromLngLat(-73.985550, 40.77900),
    Point.fromLngLat(-73.975550, 40.74859),
    Point.fromLngLat(-73.968900, 40.78859)
)

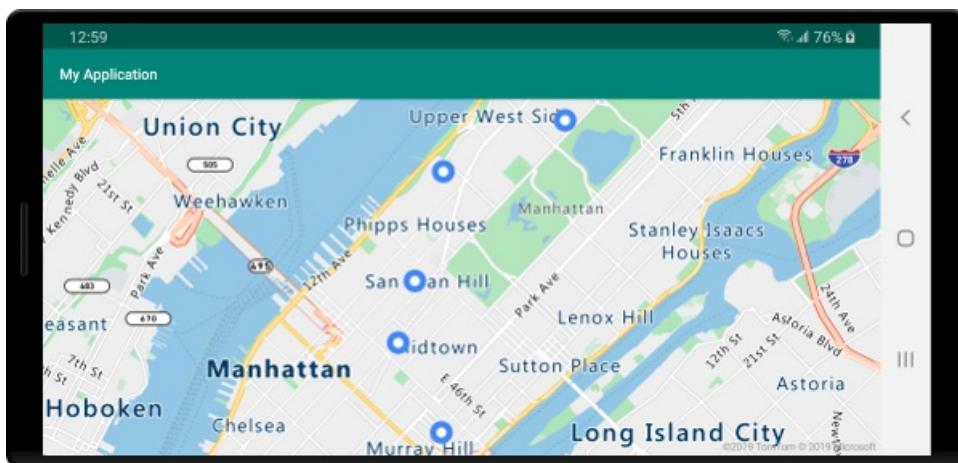
//Add multiple points to the data source.
source.add(points)

//Create a bubble layer to render the filled in area of the circle, and add it to the map.
val layer = BubbleLayer(
    source,
    bubbleRadius(5f),
    bubbleColor("white"),
    bubbleStrokeColor("#4288f7"),
    bubbleStrokeWidth(6f)
)

map.layers.add(layer)

```

The following screenshot shows the above code renders points in a bubble layer.



Show labels with a bubble layer

This code shows you how to use a bubble layer to render a point on the map. And, how to use a symbol layer to render a label. To hide the icon of the symbol layer, set the `iconImage` option to "none".

```

//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Add a data point to the map.
source.add(Point.fromLngLat(-122.336641,47.627631));

//Add a bubble layer.
map.layers.add(new BubbleLayer(source,
    bubbleRadius(5f),
    bubbleColor("white"),
    bubbleStrokeColor("#4288f7"),
    bubbleStrokeWidth(6f)
));

//Add a symbol layer to display text, hide the icon image.
map.layers.add(new SymbolLayer(source,
    //Hide the icon image.
    iconImage("none"),
    textField("Museum of History & Industry (MOHAI"),
    textColor("#005995"),
    textOffset(new Float[]{0f, -2.2f})
));

```

```

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

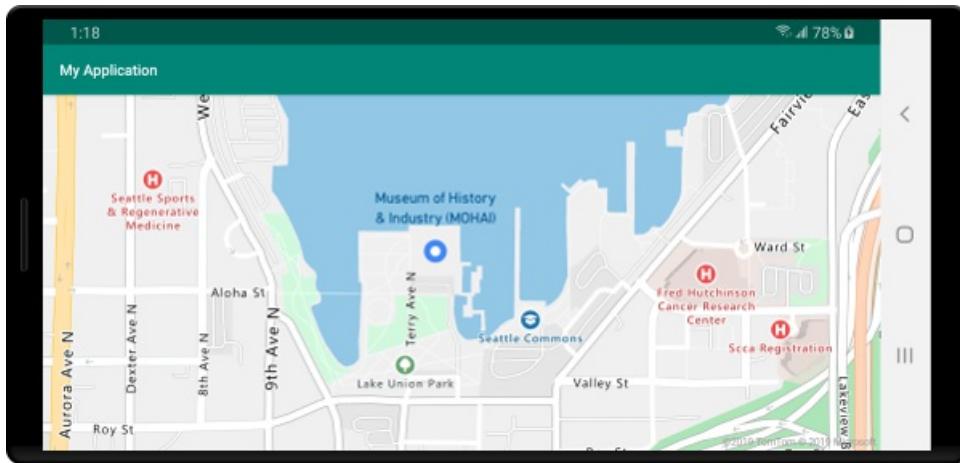
//Add a data point to the map.
source.add(Point.fromLngLat(-122.336641, 47.627631))

//Add a bubble layer.
map.layers.add(
    BubbleLayer(
        source,
        bubbleRadius(5f),
        bubbleColor("white"),
        bubbleStrokeColor("#4288f7"),
        bubbleStrokeWidth(6f)
    )
)

//Add a symbol layer to display text, hide the icon image.
map.layers.add(
    SymbolLayer(
        source, //Hide the icon image.
        iconImage("none"),
        textField("Museum of History & Industry (MOHAI"),
        textColor("#005995"),
        textOffset(arrayOf(0f, -2.2f))
    )
)

```

The following screenshot shows the above code rendering a point in a bubble layer and a text label for the point with a symbol layer.



Next steps

See the following articles for more code samples to add to your maps:

[Create a data source](#)

[Add a symbol layer](#)

[Use data-driven style expressions](#)

[Display feature information](#)

Display feature information

3/26/2021 • 6 minutes to read • [Edit Online](#)

Spatial data is often represented using points, lines, and polygons. This data often has metadata information associated with it. For example, a point may represent the location of a restaurant and metadata about that restaurant may be its name, address, and type of food it serves. This metadata can be added as properties of a GeoJSON `Feature`. The following code creates a simple point feature with a `title` property that has a value of "Hello World!"

```
//Create a data source and add it to the map.  
DataSource source = new DataSource();  
map.sources.add(source);  
  
//Create a point feature.  
Feature feature = Feature.fromGeometry(Point.fromLngLat(-122.33, 47.64));  
  
//Add a property to the feature.  
feature.addStringProperty("title", "Hello World!");  
  
//Create a point feature, pass in the metadata properties, and add it to the data source.  
source.add(feature);
```

```
//Create a data source and add it to the map.  
val source = DataSource()  
map.sources.add(source)  
  
//Create a point feature.  
val feature = Feature.fromGeometry(Point.fromLngLat(-122.33, 47.64))  
  
//Add a property to the feature.  
feature.addStringProperty("title", "Hello World!")  
  
//Create a point feature, pass in the metadata properties, and add it to the data source.  
source.add(feature)
```

See the [Create a data source](#) documentation for ways to create and add data to the map.

When a user interacts with a feature on the map, events can be used to react to those actions. A common scenario is to display a message made of the metadata properties of a feature the user interacted with. The `OnFeatureClick` event is the main event used to detect when the user tapped a feature on the map. There's also an `OnLongFeatureClick` event. When adding the `OnFeatureClick` event to the map, it can be limited to a single layer by passing in the ID of a layer to limit it to. If no layer ID is passed in, tapping any feature on the map, regardless of which layer it is in, would fire this event. The following code creates a symbol layer to render point data on the map, then adds an `OnFeatureClick` event and limits it to this symbol layer.

```

//Create a symbol and add it to the map.
SymbolLayer layer = new SymbolLayer(source);
map.layers.add(layer);

//Add a feature click event to the map.
map.events.add((OnFeatureClick) (features) -> {
    //Retrieve the title property of the feature as a string.
    String msg = features.get(0).getStringProperty("title");

    //Do something with the message.

    //Return a boolean indicating if event should be consumed or continue bubble up.
    return false;
}, layer.getId());    //Limit this event to the symbol layer.

```

```

//Create a symbol and add it to the map.
val layer = SymbolLayer(source)
map.layers.add(layer)

//Add a feature click event to the map.
map.events.add(OnFeatureClick { features: List<Feature> ->
    //Retrieve the title property of the feature as a string.
    val msg = features[0].getStringProperty("title")

    //Do something with the message.

    //Return a boolean indicating if event should be consumed or continue bubble up.
    return false
}, layer.getId()) //Limit this event to the symbol layer.

```

Display a toast message

A toast message is one of the easiest ways to display information to the user and is available in all versions of Android. It doesn't support any type of user input and is only displayed for a short period of time. If you want to quickly let the user know something about what they tapped on, a toast message might be a good option. The following code shows how a toast message can be used with the `OnFeatureClick` event.

```

//Add a feature click event to the map.
map.events.add((OnFeatureClick) (features) -> {
    //Retrieve the title property of the feature as a string.
    String msg = features.get(0).getStringProperty("title");

    //Display a toast message with the title information.
    Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();

    //Return a boolean indicating if event should be consumed or continue bubble up.
    return false;
}, layer.getId());    //Limit this event to the symbol layer.

```

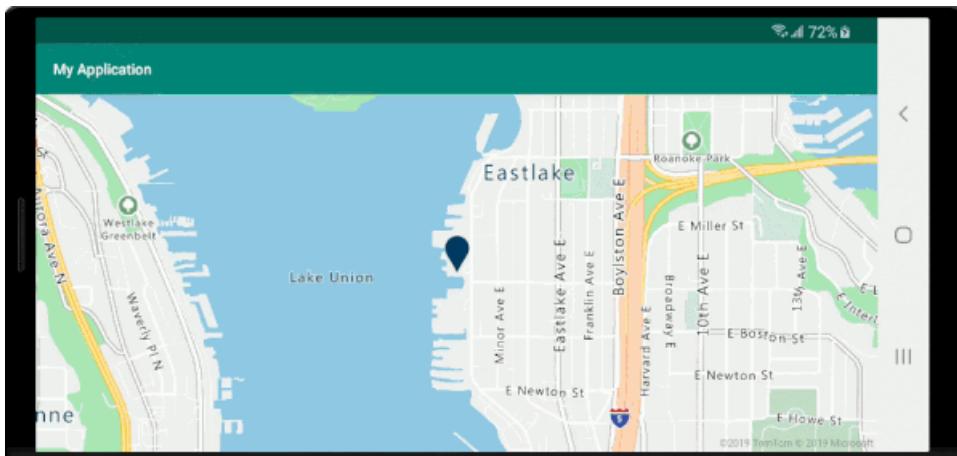
```

//Add a feature click event to the map.
map.events.add(OnFeatureClick { features: List<Feature> ->
    //Retrieve the title property of the feature as a string.
    val msg = features[0].getStringProperty("title")

    //Display a toast message with the title information.
    Toast.makeText(this, msg, Toast.LENGTH_SHORT).show()

    //Return a boolean indicating if event should be consumed or continue bubble up.
    return false
}, layer.getId()) //Limit this event to the symbol layer.

```



In addition to toast messages, There are many other ways to present the metadata properties of a feature, such as:

- [Snackbar widget](#) - [Snackbars](#) provide lightweight feedback about an operation. They show a brief message at the bottom of the screen on mobile and lower left on larger devices. [Snackbars](#) appear above all other elements on screen and only one can be displayed at a time.
- [Dialogs](#) - A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog doesn't fill the screen and is normally used for modal events that require users to take an action before they can continue.
- Add a [Fragment](#) to the current activity.
- Navigate to another activity or view.

Display a popup

The Azure Maps Android SDK provides a [Popup](#) class that makes it easy to create UI annotation elements that are anchored to a position on the map. For popups, you have to pass in a view with a relative layout into the [content](#) option of the popup. Here is a simple layout example that displays dark text on top of a white background.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:background="#ffffffff"
    android:layout_margin="8dp"
    android:padding="10dp"

    android:layout_height="match_parent">

    <TextView
        android:id="@+id/message"
        android:layout_width="wrap_content"
        android:text=""
        android:textSize="18dp"
        android:textColor="#222"
        android:layout_height="wrap_content"
        android:width="200dp"/>

</RelativeLayout>
```

Assuming the above layout is stored in a file called `popup_text.xml` in the `res -> layout` folder of an app, the following code creates a popup, adds it to the map. When a feature is clicked, the `title` property is displayed using the `popup_text.xml` layout, with the bottom center of the layout anchored to the specified position on the map.

```
//Create a popup and add it to the map.
Popup popup = new Popup();
map.popups.add(popup);

map.events.add((OnFeatureClick)(feature) -> {
    //Get the first feature and it's properties.
    Feature f = feature.get(0);
    JSONObject props = f.properties();

    //Retrieve the custom layout for the popup.
    View customView = LayoutInflater.from(this).inflate(R.layout.popup_text, null);

    //Access the text view within the custom view and set the text to the title property of the feature.
    TextView tv = customView.findViewById(R.id.message);
    tv.setText(props.get("title").getAsString());

    //Get the coordinates from the clicked feature and create a position object.
    List<Double> c = ((Point)(f.geometry())).coordinates();
    Position pos = new Position(c.get(0), c.get(1));

    //Set the options on the popup.
    popup.setOptions(
        //Set the popups position.
        position(pos),
        //Set the anchor point of the popup content.
        anchor(AnchorType.BOTTOM),
        //Set the content of the popup.
        content(customView)
        //Optionally, hide the close button of the popup.
        //, closeButton(false)
    );

    //Open the popup.
    popup.open();

    //Return a boolean indicating if event should be consumed or continue bubble up.
    return false;
});
```

```

//Create a popup and add it to the map.
val popup = Popup()
map.popups.add(popup)

map.events.add(OnFeatureClick { feature: List<Feature> ->
    //Get the first feature and its properties.
    val f = feature[0]
    val props = f.properties()

    //Retrieve the custom layout for the popup.
    val customView: View = LayoutInflater.from(this).inflate(R.layout.popup_text, null)

    //Access the text view within the custom view and set the text to the title property of the feature.
    val tv: TextView = customView.findViewById(R.id.message)
    tv.text = props!!["title"].asString

    //Get the coordinates from the clicked feature and create a position object.
    val c: List<Double> = (f.geometry() as Point?).coordinates()
    val pos = Position(c[0], c[1])

    //Set the options on the popup.
    popup.setOptions(
        //Set the popups position.
        position(pos),
        //Set the anchor point of the popup content.
        anchor(AnchorType.BOTTOM),
        //Set the content of the popup.
        content(customView)

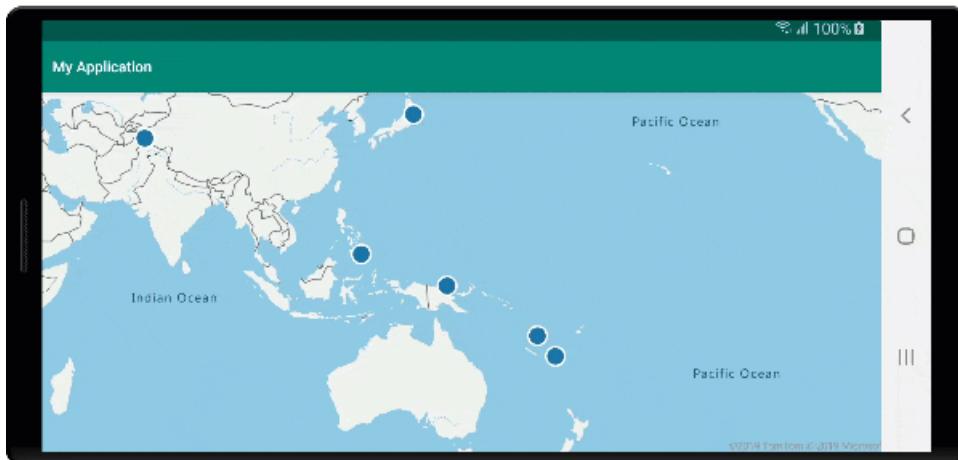
        //Optionally, hide the close button of the popup.
        //, closeButton(false)
    )

    //Open the popup.
    popup.open()

    //Return a boolean indicating if event should be consumed or continue bubble up.
    return false
})

```

The following screen capture shows popups appearing when features are clicked and staying anchored to their specified location on the map as it moves.



Next steps

To add more data to your map:

[React to map events](#)

[Create a data source](#)

[Add a symbol layer](#)

[Add a bubble layer](#)

[Add a line layer](#)

[Add a polygon layer](#)

Add a line layer to the map (Android SDK)

3/5/2021 • 7 minutes to read • [Edit Online](#)

A line layer can be used to render `LineString` and `MultiLineString` features as paths or routes on the map. A line layer can also be used to render the outline of `Polygon` and `MultiPolygon` features. A data source is connected to a line layer to provide it with data to render.

TIP

Line layers by default will render the coordinates of polygons as well as lines in a data source. To limit the layer so that it only renders `LineString` geometry features, set the `filter` option of the layer to `eq(geometryType(), "LineString")`. If you want to include `MultiLineString` features as well, set the `filter` option of the layer to `any(eq(geometryType(), "LineString"), eq(geometryType(), "MultiLineString"))`.

Prerequisites

Be sure to complete the steps in the [Quickstart: Create an Android app](#) document. Code blocks in this article can be inserted into the maps `onReady` event handler.

Add a line layer

The following code shows how to create a line. Add the line to a data source, then render it with a line layer using the `LineLayer` class.

```
//Create a data source and add it to the map.  
DataSource source = new DataSource();  
map.sources.add(source);  
  
//Create a list of points.  
List<Point> points = Arrays.asList(  
    Point.fromLngLat(-73.97234, 40.74327),  
    Point.fromLngLat(-74.00442, 40.75680));  
  
//Create a LineString geometry and add it to the data source.  
source.add(LineString.fromLngLats(points));  
  
//Create a line layer and add it to the map.  
LineLayer layer = new LineLayer(source,  
    strokeColor("blue"),  
    strokeWidth(5f)  
);  
  
map.layers.add(layer);
```

```

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

//Create a list of points.
val points = asList(
    Point.fromLngLat(-73.97234, 40.74327),
    Point.fromLngLat(-74.00442, 40.75680)
)

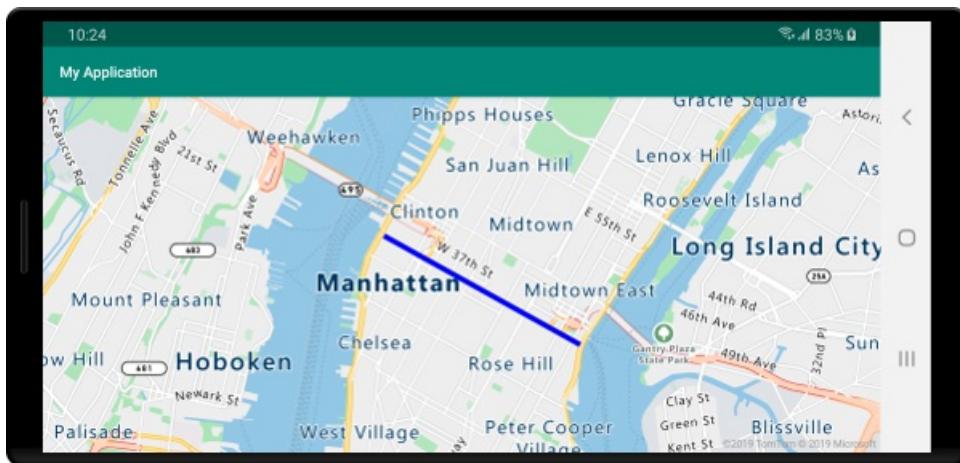
//Create a LineString geometry and add it to the data source.
source.add(LineString.fromLngLats(points))

//Create a line layer and add it to the map.
val layer = LineLayer(
    source,
    strokeColor("blue"),
    strokeWidth(5f)
)

map.layers.add(layer)

```

The following screenshot shows the above code rendering a line in a line layer.



Data-driven line style

The following code creates two line features and adds a speed limit value as a property to each line. A line layer uses a data-drive style expression color the lines based on the speed limit value. Since the line data overlays along roads, the code below adds the line layer below the label layer so that road labels can still clearly be read.

```
//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Create a line feature.
Feature feature = Feature.fromGeometry(
    LineString.fromLngLats(Arrays.asList(
        Point.fromLatLng(-122.131821, 47.704033),
        Point.fromLatLng(-122.099919, 47.703678))));

//Add a property to the feature.
feature.addNumberProperty("speedLimitMph", 35);

//Add the feature to the data source.
source.add(feature);

//Create a second line feature.
Feature feature2 = Feature.fromGeometry(
    LineString.fromLngLats(Arrays.asList(
        Point.fromLatLng(-122.126662, 47.708265),
        Point.fromLatLng(-122.126877, 47.703980))));

//Add a property to the second feature.
feature2.addNumberProperty("speedLimitMph", 15);

//Add the second feature to the data source.
source.add(feature2);

//Create a line layer and add it to the map.
LineLayer layer = new LineLayer(source,
    strokeColor(
        interpolate(
            linear(),
            get("speedLimitMph"),
            stop(0, color(Color.GREEN)),
            stop(30, color(Color.YELLOW)),
            stop(60, color(Color.RED)))
    )
),
    strokeWidth(5f)
);

map.layers.add(layer, "labels");
```

```

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

//Create a line feature.
val feature = Feature.fromGeometry(
    LineString.fromLngLats(
        Arrays.asList(
            Point.fromLngLat(-122.131821, 47.704033),
            Point.fromLngLat(-122.099919, 47.703678)
        )
    )
)

//Add a property to the feature.
feature.addNumberProperty("speedLimitMph", 35)

//Add the feature to the data source.
source.add(feature)

//Create a second line feature.
val feature2 = Feature.fromGeometry(
    LineString.fromLngLats(
        Arrays.asList(
            Point.fromLngLat(-122.126662, 47.708265),
            Point.fromLngLat(-122.126877, 47.703980)
        )
    )
)

//Add a property to the second feature.
feature2.addNumberProperty("speedLimitMph", 15)

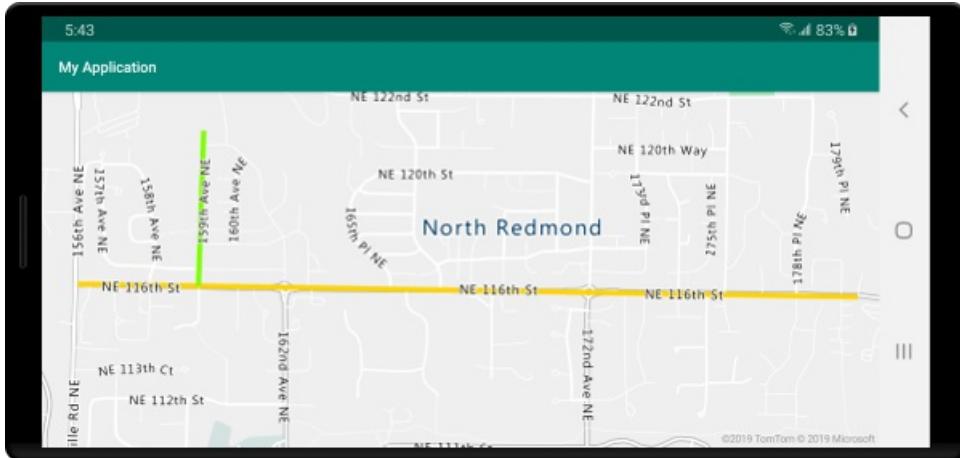
//Add the second feature to the data source.
source.add(feature2)

//Create a line layer and add it to the map.
val layer = LineLayer(
    source,
    strokeColor(
        interpolate(
            linear(),
            get("speedLimitMph"),
            stop(0, color(Color.GREEN)),
            stop(30, color(Color.YELLOW)),
            stop(60, color(Color.RED))
        )
    ),
    strokeWidth(5f)
)

map.layers.add(layer, "labels")

```

The following screenshot shows the above code rendering two lines in a line layer with their color being retrieved from a data driven style expression based on a property in the line features.



Add a stroke gradient to a line

You may apply a single stroke color to a line. You can also fill a line with a gradient of colors to show transition from one line segment to the next line segment. For example, line gradients can be used to represent changes over time and distance, or different temperatures across a connected line of objects. In order to apply this feature to a line, the data source must have the `lineMetrics` option set to `true`, and then a color gradient expression can be passed to the `strokeColor` option of the line. The stroke gradient expression has to reference the `lineProgress` data expression that exposes the calculated line metrics to the expression.

```

//Create a data source and add it to the map.
DataSource source = new DataSource(
    //Enable line metrics on the data source. This is needed to enable support for strokeGradient.
    withLineMetrics(true)
);
map.sources.add(source);

//Create a line and add it to the data source.
source.add(LineString.fromLngLats(
    Arrays.asList(
        Point.fromLngLat(-122.18822, 47.63208),
        Point.fromLngLat(-122.18204, 47.63196),
        Point.fromLngLat(-122.17243, 47.62976),
        Point.fromLngLat(-122.16419, 47.63023),
        Point.fromLngLat(-122.15852, 47.62942),
        Point.fromLngLat(-122.15183, 47.62988),
        Point.fromLngLat(-122.14256, 47.63451),
        Point.fromLngLat(-122.13483, 47.64041),
        Point.fromLngLat(-122.13466, 47.64422),
        Point.fromLngLat(-122.13844, 47.65440),
        Point.fromLngLat(-122.13277, 47.66515),
        Point.fromLngLat(-122.12779, 47.66712),
        Point.fromLngLat(-122.11595, 47.66712),
        Point.fromLngLat(-122.11063, 47.66735),
        Point.fromLngLat(-122.10668, 47.67035),
        Point.fromLngLat(-122.10565, 47.67498)
    )
));
//Create a line layer and pass in a gradient expression for the strokeGradient property.
map.layers.add(new LineLayer(source,
    strokeWidth(6f),

    //Pass an interpolate or step expression that represents a gradient.
    strokeGradient(
        interpolate(
            linear(),
            lineProgress(),
            stop(0, color(Color.BLUE)),
            stop(0.1, color(Color.argb(255, 65, 105, 225))), //Royal Blue
            stop(0.3, color(Color.CYAN)),
            stop(0.5, color(Color.argb(255,0, 255, 0))), //Lime
            stop(0.7, color(Color.YELLOW)),
            stop(1, color(Color.RED))
        )
    )
));

```

```

//Create a data source and add it to the map.
val source = DataSource(
    //Enable line metrics on the data source. This is needed to enable support for strokeGradient.
    withLineMetrics(true)
)
map.sources.add(source)

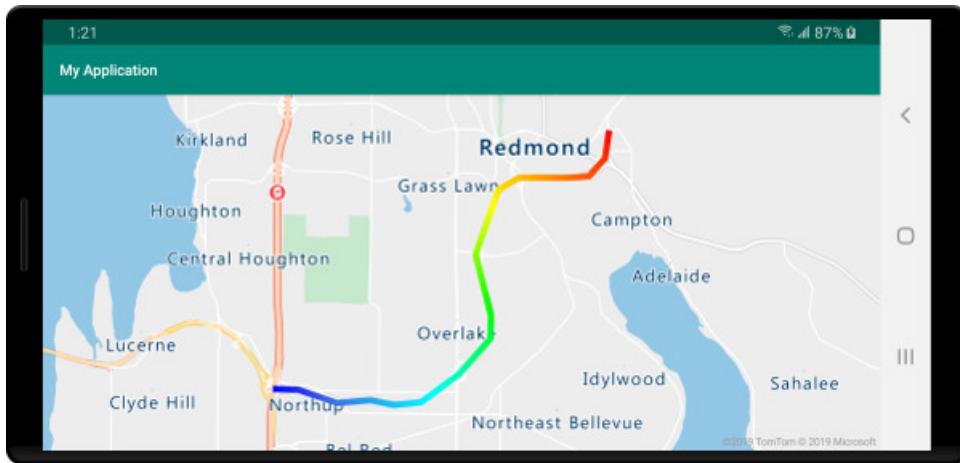
//Create a line and add it to the data source.
source.add(
    LineString.fromLngLats(
        Arrays.asList(
            Point.fromLngLat(-122.18822, 47.63208),
            Point.fromLngLat(-122.18204, 47.63196),
            Point.fromLngLat(-122.17243, 47.62976),
            Point.fromLngLat(-122.16419, 47.63023),
            Point.fromLngLat(-122.15852, 47.62942),
            Point.fromLngLat(-122.15183, 47.62988),
            Point.fromLngLat(-122.14256, 47.63451),
            Point.fromLngLat(-122.13483, 47.64041),
            Point.fromLngLat(-122.13466, 47.64422),
            Point.fromLngLat(-122.13844, 47.65440),
            Point.fromLngLat(-122.13277, 47.66515),
            Point.fromLngLat(-122.12779, 47.66712),
            Point.fromLngLat(-122.11595, 47.66712),
            Point.fromLngLat(-122.11063, 47.66735),
            Point.fromLngLat(-122.10668, 47.67035),
            Point.fromLngLat(-122.10565, 47.67498)
        )
    )
)

//Create a line layer and pass in a gradient expression for the strokeGradient property.
map.layers.add(
    LineLayer(
        source,
        strokeWidth(6f),

        //Pass an interpolate or step expression that represents a gradient.
        strokeGradient(
            interpolate(
                linear(),
                lineProgress(),
                stop(0, color(Color.BLUE)),
                stop(0.1, color(Color.argb(255, 65, 105, 225))), //Royal Blue
                stop(0.3, color(Color.CYAN)),
                stop(0.5, color(Color.argb(255, 0, 255, 0))), //Lime
                stop(0.7, color(Color.YELLOW)),
                stop(1, color(Color.RED))
            )
        )
    )
)

```

The following screenshot shows the above code displaying a line rendered using a gradient stroke color.



Add symbols along a line

This sample shows how to add arrow icons along a line on the map. When using a symbol layer, set the `symbolPlacement` option to `SymbolPlacement.LINE`. This option will render the symbols along the line and rotate the icons (0 degrees = right).

```
//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Load a image of an arrow into the map image sprite and call it "arrow-icon".
map.images.add("arrow-icon", R.drawable.purple_arrow_right);

//Create and add a line to the data source.
source.add(LineString.fromLngLats(Arrays.asList(
    Point.fromLngLat(-122.18822, 47.63208),
    Point.fromLngLat(-122.18204, 47.63196),
    Point.fromLngLat(-122.17243, 47.62976),
    Point.fromLngLat(-122.16419, 47.63023),
    Point.fromLngLat(-122.15852, 47.62942),
    Point.fromLngLat(-122.15183, 47.62988),
    Point.fromLngLat(-122.14256, 47.63451),
    Point.fromLngLat(-122.13483, 47.64041),
    Point.fromLngLat(-122.13466, 47.64422),
    Point.fromLngLat(-122.13844, 47.65440),
    Point.fromLngLat(-122.13277, 47.66515),
    Point.fromLngLat(-122.12779, 47.66712),
    Point.fromLngLat(-122.11595, 47.66712),
    Point.fromLngLat(-122.11063, 47.66735),
    Point.fromLngLat(-122.10668, 47.67035),
    Point.fromLngLat(-122.10565, 47.67498)))
));

//Create a line layer and add it to the map.
map.layers.add(new LineLayer(source,
    strokeColor("DarkOrchid"),
    strokeWidth(5f)
));

//Create a symbol layer and add it to the map.
map.layers.add(new SymbolLayer(source,
    //Space symbols out along line.
    symbolPlacement(SymbolPlacement.LINE),

    //Spread the symbols out 100 pixels apart.
    symbolSpacing(100f),

    //Use the arrow icon as the symbol.
    iconImage("arrow-icon"),

    //Allow icons to overlap so that they aren't hidden if they collide with other map elements.
    iconAllowOverlap(true),

    //Center the symbol icon.
    iconAnchor(AnchorType.CENTER),

    //Scale the icon size.
    iconSize(0.8f)
));
```

```

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

//Load a image of an arrow into the map image sprite and call it "arrow-icon".
map.images.add("arrow-icon", R.drawable.purple_arrow_right)

//Create and add a line to the data source.
//Create and add a line to the data source.
source.add(
    LineString.fromLngLats(
        Arrays.asList(
            Point.fromLngLat(-122.18822, 47.63208),
            Point.fromLngLat(-122.18204, 47.63196),
            Point.fromLngLat(-122.17243, 47.62976),
            Point.fromLngLat(-122.16419, 47.63023),
            Point.fromLngLat(-122.15852, 47.62942),
            Point.fromLngLat(-122.15183, 47.62988),
            Point.fromLngLat(-122.14256, 47.63451),
            Point.fromLngLat(-122.13483, 47.64041),
            Point.fromLngLat(-122.13466, 47.64422),
            Point.fromLngLat(-122.13844, 47.65440),
            Point.fromLngLat(-122.13277, 47.66515),
            Point.fromLngLat(-122.12779, 47.66712),
            Point.fromLngLat(-122.11595, 47.66712),
            Point.fromLngLat(-122.11063, 47.66735),
            Point.fromLngLat(-122.10668, 47.67035),
            Point.fromLngLat(-122.10565, 47.67498)
        )
    )
)

//Create a line layer and add it to the map.
map.layers.add(
    LineLayer(
        source,
        strokeColor("DarkOrchid"),
        strokeWidth(5f)
    )
)

//Create a symbol layer and add it to the map.
map.layers.add(
    SymbolLayer(
        source, //Space symbols out along line.
        symbolPlacement(SymbolPlacement.LINE), //Spread the symbols out 100 pixels apart.
        symbolSpacing(100f), //Use the arrow icon as the symbol.
        iconImage("arrow-icon"), //Allow icons to overlap so that they aren't hidden if they collide with other map elements.
        iconAllowOverlap(true), //Center the symbol icon.
        iconAnchor(AnchorType.CENTER), //Scale the icon size.
        iconSize(0.8f)
    )
)

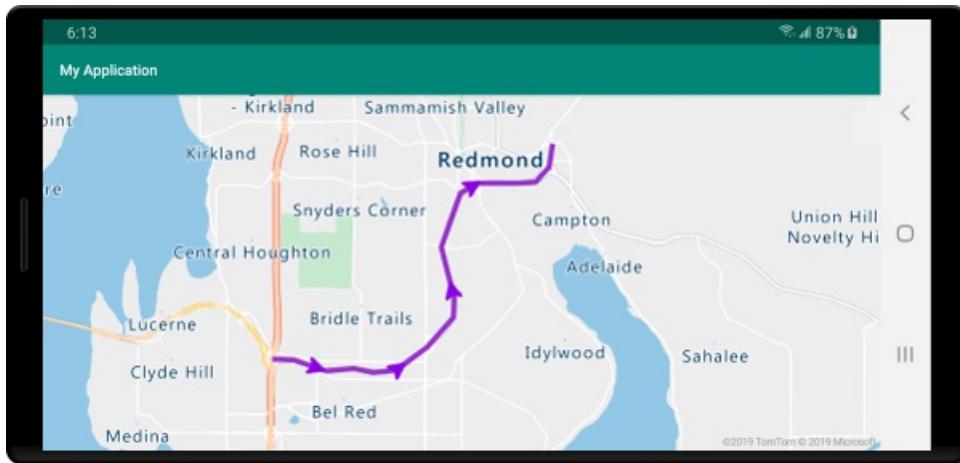
```

For this sample, the following image was loaded into the drawable folder of the app.



`purple_arrow_right.png`

The screenshot below shows the above code displaying a line with arrow icons displayed along it.



Next steps

See the following articles for more code samples to add to your maps:

[Create a data source](#)

[Use data-driven style expressions](#)

[Add a polygon layer](#)

Add a polygon layer to the map (Android SDK)

3/5/2021 • 3 minutes to read • [Edit Online](#)

This article shows you how to render the areas of `Polygon` and `MultiPolygon` feature geometries on the map using a polygon layer.

Prerequisites

Be sure to complete the steps in the [Quickstart: Create an Android app](#) document. Code blocks in this article can be inserted into the maps `onReady` event handler.

Use a polygon layer

When a polygon layer is connected to a data source and loaded on the map, it renders the area with `Polygon` and `MultiPolygon` features. To create a polygon, add it to a data source, and render it with a polygon layer using the `PolygonLayer` class.

```
//Create a data source and add it to the map.  
DataSource source = new DataSource();  
map.sources.add(source);  
  
//Create a rectangular polygon.  
source.add(Polygon.fromLngLats(  
    Arrays.asList(  
        Arrays.asList(  
            Point.fromLngLat(-73.98235, 40.76799),  
            Point.fromLngLat(-73.95785, 40.80044),  
            Point.fromLngLat(-73.94928, 40.79680),  
            Point.fromLngLat(-73.97317, 40.76437),  
            Point.fromLngLat(-73.98235, 40.76799)  
        )  
    )  
));  
  
//Create and add a polygon layer to render the polygon on the map, below the label layer.  
map.layers.add(new PolygonLayer(source,  
    fillColor("red"),  
    fillOpacity(0.7f)  
, "labels");
```

```

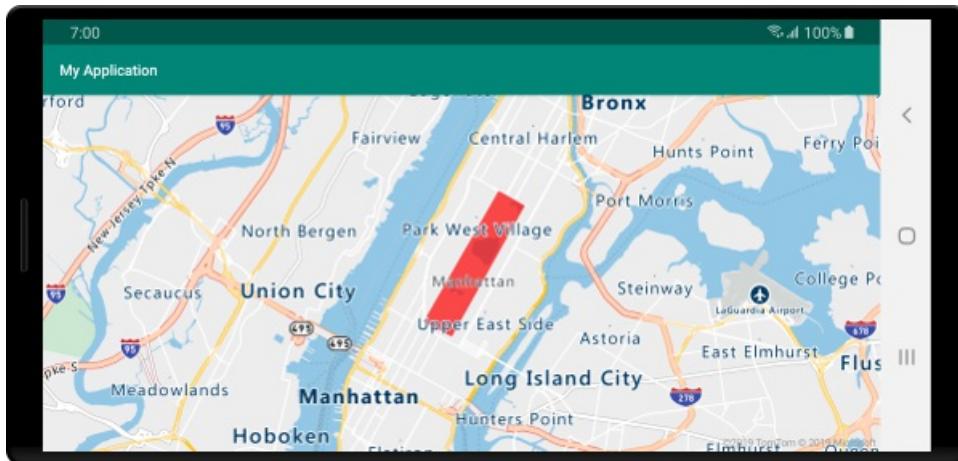
//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

//Create a rectangular polygon.
source.add(
    Polygon.fromLngLats(
        Arrays.asList(
            Arrays.asList(
                Point.fromLngLat(-73.98235, 40.76799),
                Point.fromLngLat(-73.95785, 40.80044),
                Point.fromLngLat(-73.94928, 40.79680),
                Point.fromLngLat(-73.97317, 40.76437),
                Point.fromLngLat(-73.98235, 40.76799)
            )
        )
    )
)

//Create and add a polygon layer to render the polygon on the map, below the label layer.
map.layers.add(
    PolygonLayer(
        source,
        fillColor("red"),
        fillOpacity(0.7f)
    ), "labels"
)

```

The following screenshot shows the above code rendering the area of a polygon using a polygon layer.



Use a polygon and line layer together

A line layer is used to render the outline of polygons. The following code sample renders a polygon like the previous example, but now adds a line layer. This line layer is a second layer connected to the data source.

```

//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Create a rectangular polygon.
source.add(Polygon.fromLngLats(
    Arrays.asList(
        Arrays.asList(
            Point.fromLngLat(-73.98235, 40.76799),
            Point.fromLngLat(-73.95785, 40.80044),
            Point.fromLngLat(-73.94928, 40.79680),
            Point.fromLngLat(-73.97317, 40.76437),
            Point.fromLngLat(-73.98235, 40.76799)
        )
    )
));

//Create and add a polygon layer to render the polygon on the map, below the label layer.
map.layers.add(new PolygonLayer(source,
    fillColor("rgba(0, 200, 200, 0.5)")
), "labels");

//Create and add a line layer to render the outline of the polygon.
map.layers.add(new LineLayer(source,
    strokeColor("red"),
    strokeWidth(2f)
));

```

```

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

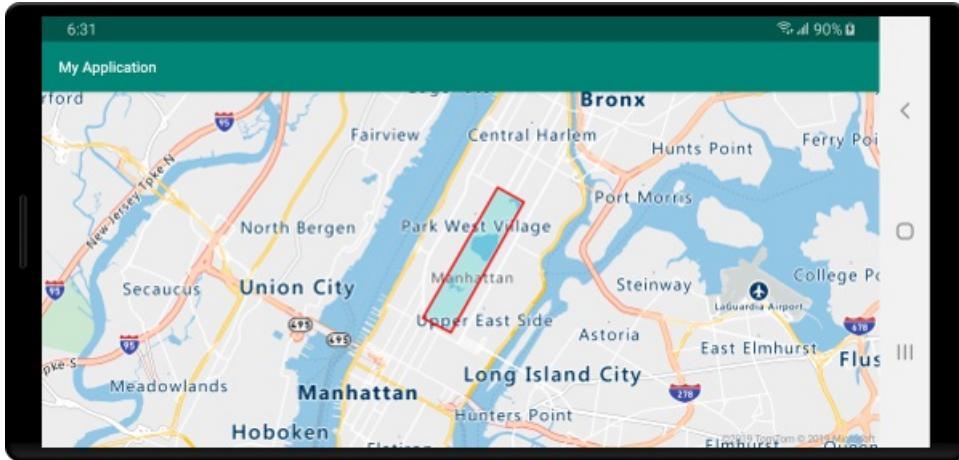
//Create a rectangular polygon.
source.add(
    Polygon.fromLngLats(
        Arrays.asList(
            Arrays.asList(
                Point.fromLngLat(-73.98235, 40.76799),
                Point.fromLngLat(-73.95785, 40.80044),
                Point.fromLngLat(-73.94928, 40.79680),
                Point.fromLngLat(-73.97317, 40.76437),
                Point.fromLngLat(-73.98235, 40.76799)
            )
        )
    )
)

//Create and add a polygon layer to render the polygon on the map, below the label layer.
map.layers.add(
    PolygonLayer(
        source,
        fillColor("rgba(0, 200, 200, 0.5)")
    ), "labels"
)

//Create and add a line layer to render the outline of the polygon.
map.layers.add(
    LineLayer(
        source,
        strokeColor("red"),
        strokeWidth(2f)
    )
)

```

The following screenshot shows the above code rendering a polygon with its outline rendered using a line layer.



TIP

When outlining a polygon with a line layer, be sure to close all rings in polygons such that each array of points has the same start and end point. If this is not done, the line layer may not connect the last point of the polygon to the first point.

Fill a polygon with a pattern

In addition to filling a polygon with a color, you may use an image pattern to fill the polygon. Load an image pattern into the maps image sprite resources and then reference this image with the `fillPattern` option of the polygon layer.

```
//Load an image pattern into the map image sprite.  
map.images.add("fill-checker-red", R.drawable.fill_checker_red);  
  
//Create a data source and add it to the map.  
DataSource source = new DataSource();  
map.sources.add(source);  
  
//Create a polygon.  
source.add(Polygon.fromLngLats(  
    Arrays.asList(  
        Arrays.asList(  
            Point.fromLatLng(-50, -20),  
            Point.fromLatLng(0, 40),  
            Point.fromLatLng(50, -20),  
            Point.fromLatLng(-50, -20)  
        )  
    )  
));  
  
//Create and add a polygon layer to render the polygon on the map, below the label layer.  
map.layers.add(new PolygonLayer(source,  
    fillPattern("fill-checker-red"),  
    fillOpacity(0.5f)  
, "labels");
```

```

//Load an image pattern into the map image sprite.
map.images.add("fill-checker-red", R.drawable.fill_checker_red)

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

//Create a polygon.
source.add(
    Polygon.fromLngLats(
        Arrays.asList(
            Arrays.asList(
                Point.fromLngLat(-50, -20),
                Point.fromLngLat(0, 40),
                Point.fromLngLat(50, -20),
                Point.fromLngLat(-50, -20)
            )
        )
    )
)

//Create and add a polygon layer to render the polygon on the map, below the label layer.
map.layers.add(
    PolygonLayer(
        source,
        fillPattern("fill-checker-red"),
        fillOpacity(0.5f)
    ), "labels"
)

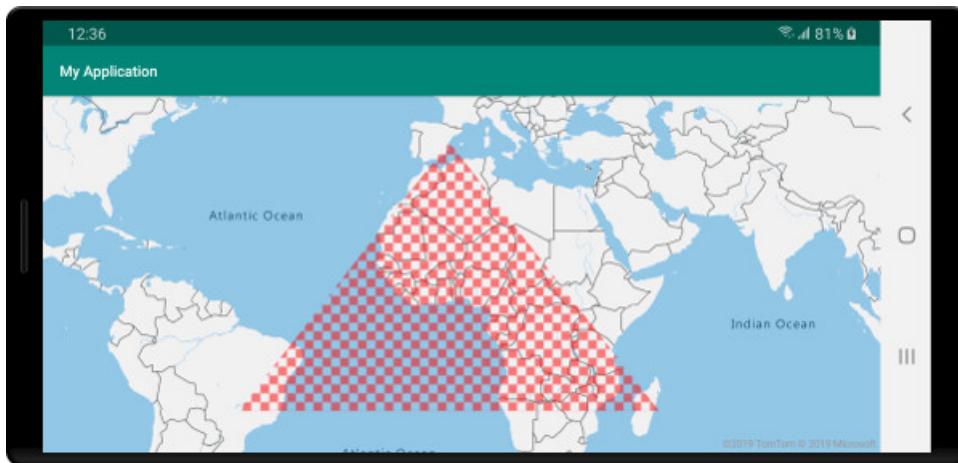
```

For this sample, the following image was loaded into the drawable folder of the app.



fill_checker_red.png

The following is a screenshot of the above code rendering a polygon with a fill pattern on the map.



Next steps

See the following articles for more code samples to add to your maps:

[Create a data source](#)

[Use data-driven style expressions](#)

[Add a line layer](#)

Add a polygon extrusion layer

Add a polygon extrusion layer to the map (Android SDK)

3/26/2021 • 3 minutes to read • [Edit Online](#)

This article shows you how to use the polygon extrusion layer to render areas of `Polygon` and `MultiPolygon` feature geometries as extruded shapes.

Use a polygon extrusion layer

Connect the polygon extrusion layer to a data source. Then, load it on the map. The polygon extrusion layer renders the areas of a `Polygon` and `MultiPolygon` features as extruded shapes. The `height` and `base` properties of the polygon extrusion layer define the base distance from the ground and height of the extruded shape in **meters**. The following code shows how to create a polygon, add it to a data source, and render it using the Polygon extrusion layer class.

NOTE

The `base` value defined in the polygon extrusion layer should be less than or equal to that of the `height`.

```
//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Create a polygon.
source.add(Polygon.fromLngLats(
    Arrays.asList(
        Arrays.asList(
            Point.fromLngLat(-73.958383, 40.800279),
            Point.fromLngLat(-73.981547, 40.768459),
            Point.fromLngLat(-73.981246, 40.767761),
            Point.fromLngLat(-73.973618, 40.764616),
            Point.fromLngLat(-73.973060, 40.765128),
            Point.fromLngLat(-73.972599, 40.764908),
            Point.fromLngLat(-73.949446, 40.796584),
            Point.fromLngLat(-73.949661, 40.797088),
            Point.fromLngLat(-73.957815, 40.800523),
            Point.fromLngLat(-73.958383, 40.800279)
        )
    )
));

//Create and add a polygon extrusion layer to the map below the labels so that they are still readable.
PolygonExtrusionLayer layer = new PolygonExtrusionLayer(source,
    fillColor("#fc0303"),
    fillOpacity(0.7f),
    height(500f)
);

//Create and add a polygon extrusion layer to the map below the labels so that they are still readable.
map.layers.add(layer, "labels");
```

```

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

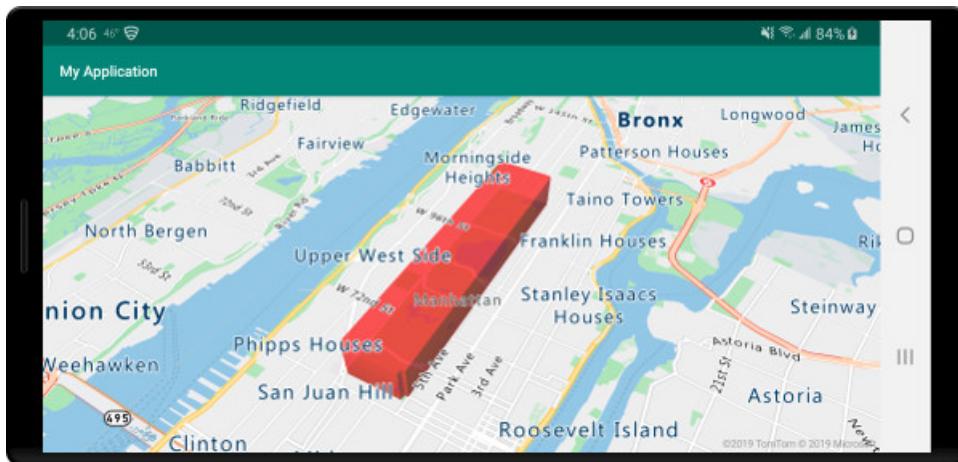
//Create a polygon.
source.add(
    Polygon.fromLngLats(
        Arrays.asList(
            Arrays.asList(
                Point.fromLngLat(-73.958383, 40.800279),
                Point.fromLngLat(-73.981547, 40.768459),
                Point.fromLngLat(-73.981246, 40.767761),
                Point.fromLngLat(-73.973618, 40.764616),
                Point.fromLngLat(-73.973060, 40.765128),
                Point.fromLngLat(-73.972599, 40.764908),
                Point.fromLngLat(-73.949446, 40.796584),
                Point.fromLngLat(-73.949661, 40.797088),
                Point.fromLngLat(-73.957815, 40.800523),
                Point.fromLngLat(-73.958383, 40.800279)
            )
        )
    )
)

//Create and add a polygon extrusion layer to the map below the labels so that they are still readable.
val layer = PolygonExtrusionLayer(
    source,
    fillColor("#fc0303"),
    fillOpacity(0.7f),
    height(500f)
)

//Create and add a polygon extrusion layer to the map below the labels so that they are still readable.
map.layers.add(layer, "labels")

```

The following screenshot shows the above code rendering a polygon stretched vertically using a polygon extrusion layer.



Add data driven polygons

A choropleth map can be rendered using the polygon extrusion layer. Set the `height` and `fillColor` properties of the extrusion layer to the measurement of the statistical variable in the `Polygon` and `MultiPolygon` feature geometries. The following code sample shows an extruded choropleth map of the United States based on the measurement of the population density by state.

```
//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Import the geojson data and add it to the data source.
Utils.importData("https://azurereadystoresamples.azurewebsites.net/Common/data/geojson/US_States_Population_De
nsity.json", this, (String result) -> {
    //Parse the data as a GeoJSON Feature Collection.
    FeatureCollection fc = FeatureCollection.fromJson(result);

    //Add the feature collection to the data source.
    source.add(fc);
});

//Create and add a polygon extrusion layer to the map below the labels so that they are still readable.
PolygonExtrusionLayer layer = new PolygonExtrusionLayer(source,
    fillOpacity(0.7f),
    fillColor(
        step(
            get("density"),
            literal("rgba(0, 255, 128, 1)"),
            stop(10, "rgba(9, 224, 118, 1)"),
            stop(20, "rgba(11, 191, 103, 1)"),
            stop(50, "rgba(247, 227, 5, 1)"),
            stop(100, "rgba(247, 199, 7, 1)"),
            stop(200, "rgba(247, 130, 5, 1)"),
            stop(500, "rgba(247, 94, 5, 1)"),
            stop(1000, "rgba(247, 37, 5, 1)")
        )
    ),
    height(
        interpolate(
            linear(),
            get("density"),
            stop(0, 100),
            stop(1200, 960000)
        )
    )
);

//Create and add a polygon extrusion layer to the map below the labels so that they are still readable.
map.layers.add(layer, "labels");
```

```

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

//Import the geojson data and add it to the data source.
Utils.importData("https://azurermapscodeexamples.azurewebsites.net/Common/data/geojson/US_States_Population_De
nsity.json",
    this
) { result: String? ->
    //Parse the data as a GeoJSON Feature Collection.
    val fc = FeatureCollection.fromJson(result!!)

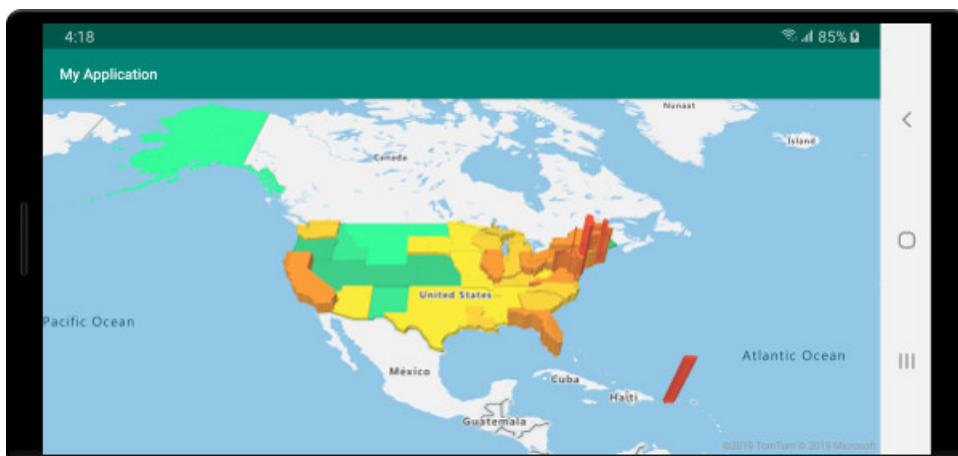
    //Add the feature collection to the data source.
    source.add(fc)
}

//Create and add a polygon extrusion layer to the map below the labels so that they are still readable.
val layer = PolygonExtrusionLayer(
    source,
    fillOpacity(0.7f),
    fillColor(
        step(
            get("density"),
            literal("rgba(0, 255, 128, 1)"),
            stop(10, "rgba(9, 224, 118, 1)"),
            stop(20, "rgba(11, 191, 103, 1)"),
            stop(50, "rgba(247, 227, 5, 1)"),
            stop(100, "rgba(247, 199, 7, 1)"),
            stop(200, "rgba(247, 130, 5, 1)"),
            stop(500, "rgba(247, 94, 5, 1)"),
            stop(1000, "rgba(247, 37, 5, 1)")
        )
    ),
    height(
        interpolate(
            linear(),
            get("density"),
            stop(0, 100),
            stop(1200, 960000)
        )
    )
)

//Create and add a polygon extrusion layer to the map below the labels so that they are still readable.
map.layers.add(layer, "labels")

```

The following screenshot shows a choropleth map of US states colored and stretched vertically as extruded polygons based on population density.



Next steps

See the following articles for more code samples to add to your maps:

[Create a data source](#)

[Use data-driven style expressions](#)

[Add a polygon layer](#)

Add a heat map layer (Android SDK)

3/5/2021 • 8 minutes to read • [Edit Online](#)

Heat maps, also known as point density maps, are a type of data visualization. They're used to represent the density of data using a range of colors and show the data "hot spots" on a map. Heat maps are a great way to render datasets with large number of points.

Rendering tens of thousands of points as symbols can cover most of the map area. This case likely results in many symbols overlapping. Making it difficult to gain a better understanding of the data. However, visualizing this same dataset as a heat map makes it easy to see the density and the relative density of each data point.

You can use heat maps in many different scenarios, including:

- **Temperature data:** Provides approximations for what the temperature is between two data points.
- **Data for noise sensors:** Shows not only the intensity of the noise where the sensor is, but it can also provide insight into the dissipation over a distance. The noise level at any one site might not be high. If the noise coverage area from multiple sensors overlaps, it's possible that this overlapping area might experience higher noise levels. As such, the overlapped area would be visible in the heat map.
- **GPS trace:** Includes the speed as a weighted height map, where the intensity of each data point is based on the speed. For example, this functionality provides a way to see where a vehicle was speeding.

TIP

Heat map layers by default render the coordinates of all geometries in a data source. To limit the layer so that it only renders point geometry features, set the `filter` option of the layer to `eq(geometryType(), "Point")`. If you want to include MultiPoint features as well, set the `filter` option of the layer to `any(eq(geometryType(), "Point"), eq(geometryType(), "MultiPoint"))`.

Prerequisites

Be sure to complete the steps in the [Quickstart: Create an Android app](#) document. Code blocks in this article can be inserted into the maps `onReady` event handler.

Add a heat map layer

To render a data source of points as a heat map, pass your data source into an instance of the `HeatMapLayer` class, and add it to the map.

The following code sample loads a GeoJSON feed of earthquakes from the past week and renders them as a heat map. Each data point is rendered with a radius of 10 pixels at all zoom levels. To ensure a better user experience, the heat map is below the label layer so the labels stay clearly visible. The data in this sample is from the [USGS Earthquake Hazards Program](#). This sample loads GeoJSON data from the web using the data import utility code block provided in the [Create a data source](#) document.

```
//Create a data source and add it to the map.
DataSource source = new DataSource();
map.sources.add(source);

//Create a heat map layer.
HeatMapLayer layer = new HeatMapLayer(source,
    heatmapRadius(10f),
    heatmapOpacity(0.8f)
);

//Add the layer to the map, below the labels.
map.layers.add(layer, "labels");

//Import the geojson data and add it to the data source.
Utils.importData("https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.geojson",
    this,
    (String result) -> {
        //Parse the data as a GeoJSON Feature Collection.
        FeatureCollection fc = FeatureCollection.fromJson(result);

        //Add the feature collection to the data source.
        source.add(fc);

        //Optionally, update the maps camera to focus in on the data.

        //Calculate the bounding box of all the data in the Feature Collection.
        BoundingBox bbox = MapMath.fromData(fc);

        //Update the maps camera so it is focused on the data.
        map.setCamera(
            bounds(bbox),
            padding(20));
    });
});
```

```

//Create a data source and add it to the map.
val source = DataSource()
map.sources.add(source)

//Create a heat map layer.
val layer = HeatMapLayer(
    source,
    heatmapRadius(10f),
    heatmapOpacity(0.8f)
)

//Add the layer to the map, below the labels.
map.layers.add(layer, "labels")

//Import the geojson data and add it to the data source.
Utils.importData("https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_week.geojson",
    this
) { result: String? ->
    //Parse the data as a GeoJSON Feature Collection.
    val fc = FeatureCollection.fromJson(result!!)

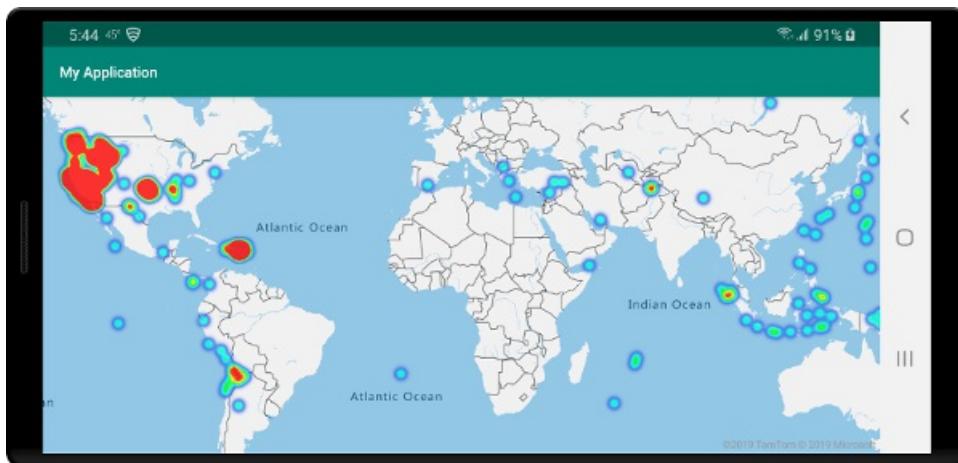
    //Add the feature collection to the data source.
    source.add(fc)

    //Optionally, update the maps camera to focus in on the data.
    //Calculate the bounding box of all the data in the Feature Collection.
    val bbox = MapMath.fromData(fc)

    //Update the maps camera so it is focused on the data.
    map.setCamera(
        bounds(bbox),
        padding(20)
    )
}

```

The following screenshot shows a map loading a heat map using the above code.



Customize the heat map layer

The previous example customized the heat map by setting the radius and opacity options. The heat map layer provides several options for customization, including:

- `heatmapRadius` : Defines a pixel radius in which to render each data point. You can set the radius as a fixed number or as an expression. By using an expression, you can scale the radius based on the zoom level, and represent a consistent spatial area on the map (for example, a 5-mile radius).
- `heatmapColor` : Specifies how the heat map is colorized. A color gradient is a common feature of heat maps. You can achieve the effect with an `interpolate` expression. You can also use a `step` expression for

colorizing the heat map, breaking up the density visually into ranges that resemble a contour or radar style map. These color palettes define the colors from the minimum to the maximum density value.

You specify color values for heat maps as an expression on the `heatmapDensity` value. The color of area where there's no data is defined at index 0 of the "Interpolation" expression, or the default color of a "Stepped" expression. You can use this value to define a background color. Often, this value is set to transparent, or a semi-transparent black.

Here are examples of color expressions:

INTERPOLATION COLOR EXPRESSION	STEPPED COLOR EXPRESSION
<pre>interpolate(linear(), heatmapDensity(), stop(0, color(Color.TRANSPARENT)), stop(0.01, color(Color:MAGENTA)), stop(0.5, color(parseColor("#fb00fb"))), stop(1, color(parseColor("#00c3ff"))))</pre>	<pre>step(heatmapDensity(), color(Color.TRANSPARENT), stop(0.01, color(parseColor("#000080"))), stop(0.25, color(parseColor("#000080"))), stop(0.5, color(Color.GREEN)), stop(0.5, color(Color.YELLOW)), stop(1, color(Color.RED)))</pre>

- `heatmapOpacity` : Specifies how opaque or transparent the heat map layer is.
- `heatmapIntensity` : Applies a multiplier to the weight of each data point to increase the overall intensity of the heatmap. It causes a difference in the weight of data points, making it easier to visualize.
- `heatmapWeight` : By default, all data points have a weight of 1, and are weighted equally. The weight option acts as a multiplier, and you can set it as a number or an expression. If a number is set as the weight, it's the equivalence of placing each data point on the map twice. For instance, if the weight is `2`, then the density doubles. Setting the weight option to a number renders the heat map in a similar way to using the intensity option.

However, if you use an expression, the weight of each data point can be based on the properties of each data point. For example, suppose each data point represents an earthquake. The magnitude value has been an important metric for each earthquake data point. Earthquakes happen all the time, but most have a low magnitude, and aren't noticed. Use the magnitude value in an expression to assign the weight to each data point. By using the magnitude value to assign the weight, you get a better representation of the significance of earthquakes within the heat map.

- `minZoom` and `maxZoom` : The zoom level range where the layer should be displayed.
- `filter` : A filter expression used to limit the retrieved from the source and rendered in the layer.
- `sourceLayer` : If the data source connected to the layer is a vector tile source, a source layer within the vector tiles must be specified.
- `visible` : Hides or shows the layer.

This following is an example of a heat map where a liner interpolation expression is used to create a smooth color gradient. The `mag` property defined in the data is used with an exponential interpolation to set the weight or relevance of each data point.

```

HeatMapLayer layer = new HeatMapLayer(source,
    heatmapRadius(10f),

    //A linear interpolation is used to create a smooth color gradient based on the heat map density.
    heatmapColor(
        interpolate(
            linear(),
            heatmapDensity(),
            stop(0, color(Color.TRANSPARENT)),
            stop(0.01, color(Color.BLACK)),
            stop(0.25, color(Color.MAGENTA)),
            stop(0.5, color(Color.RED)),
            stop(0.75, color(Color.YELLOW)),
            stop(1, color(Color.WHITE))
        )
    ),
    //Using an exponential interpolation since earthquake magnitudes are on an exponential scale.
    heatmapWeight(
        interpolate(
            exponential(2),
            get("mag"),
            stop(0,0),
            //Any earthquake above a magnitude of 6 will have a weight of 1
            stop(6, 1)
        )
    )
);

```

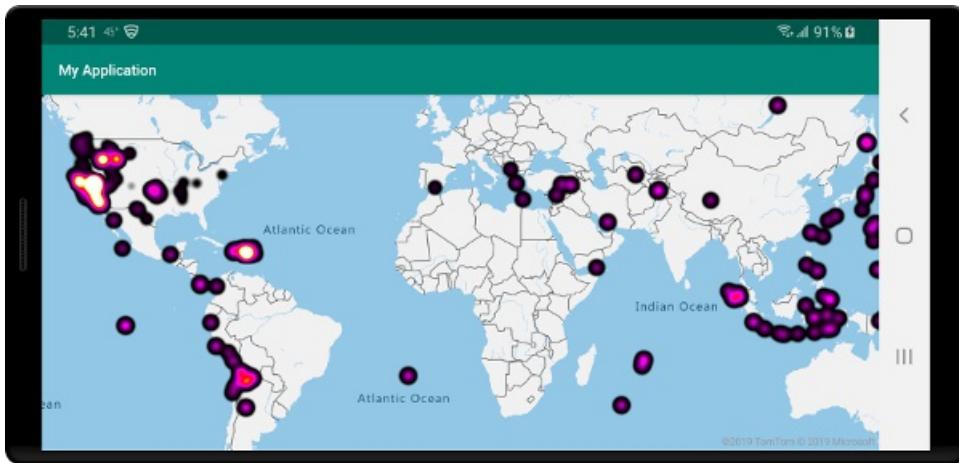
```

val layer = HeatMapLayer(source,
    heatmapRadius(10f),

    //A linear interpolation is used to create a smooth color gradient based on the heat map density.
    heatmapColor(
        interpolate(
            linear(),
            heatmapDensity(),
            stop(0, color(Color.TRANSPARENT)),
            stop(0.01, color(Color.BLACK)),
            stop(0.25, color(Color.MAGENTA)),
            stop(0.5, color(Color.RED)),
            stop(0.75, color(Color.YELLOW)),
            stop(1, color(Color.WHITE))
        )
    ),
    //Using an exponential interpolation since earthquake magnitudes are on an exponential scale.
    heatmapWeight(
        interpolate(
            exponential(2),
            get("mag"),
            stop(0,0),
            //Any earthquake above a magnitude of 6 will have a weight of 1
            stop(6, 1)
        )
    )
)

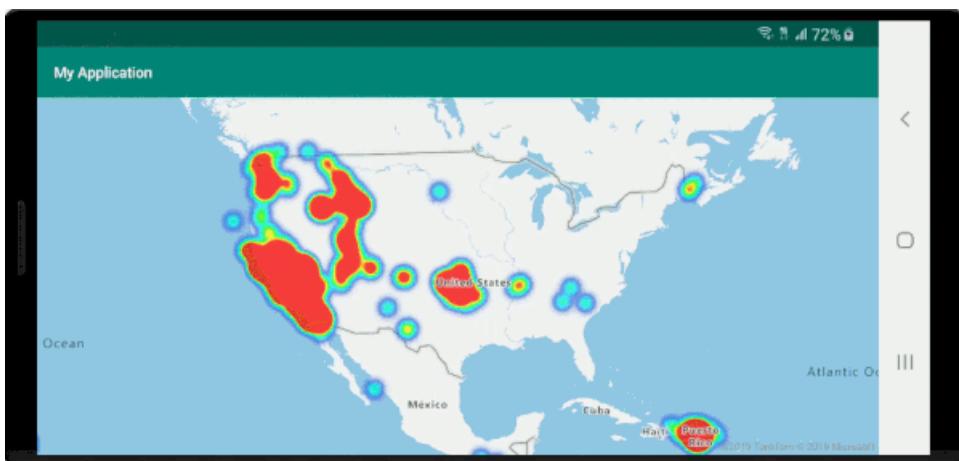
```

The following screenshot shows the above custom heat map layer using the same data from the previous heat map example.



Consistent zoomable heat map

By default, the radii of data points rendered in the heat map layer have a fixed pixel radius for all zoom levels. As you zoom the map, the data aggregates together and the heat map layer looks different. The following video shows the default behavior of the heat map where it maintains a pixel radius when zooming the map.



Use a `zoom` expression to scale the radius for each zoom level, such that each data point covers the same physical area of the map. This expression makes the heat map layer look more static and consistent. Each zoom level of the map has twice as many pixels vertically and horizontally as the previous zoom level.

Scaling the radius so that it doubles with each zoom level creates a heat map that looks consistent on all zoom levels. To apply this scaling, use `zoom` with a base 2 `exponential interpolation` expression, with the pixel radius set for the minimum zoom level and a scaled radius for the maximum zoom level calculated as

`2 * Math.pow(2, minZoom - maxZoom)` as shown in the following sample. Zoom the map to see how the heat map scales with the zoom level.

```

HeatMapLayer layer = new HeatMapLayer(source,
    heatmapRadius(
        interpolate(
            exponential(2),
            zoom(),
            //For zoom level 1 set the radius to 2 pixels.
            stop(1, 2f),
            //Between zoom level 1 and 19, exponentially scale the radius from 2 pixels to 2 * (maxZoom - minZoom)^2 pixels.
            stop(19, Math.pow(2, 19 - 1) * 2f)
        )
    ),
    heatmapOpacity(0.75f)
);

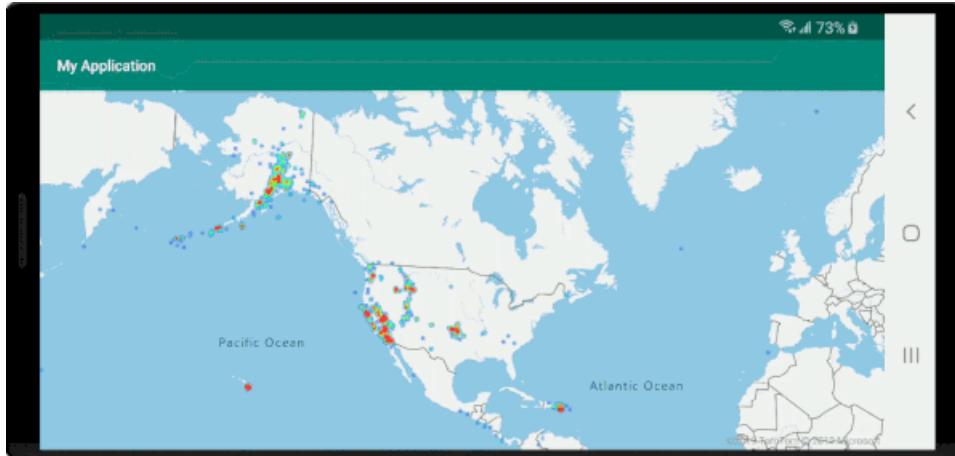
```

```

val layer = HeatMapLayer(source,
    heatmapRadius(
        interpolate(
            exponential(2),
            zoom(),
            //For zoom level 1 set the radius to 2 pixels.
            stop(1, 2f),
            //Between zoom level 1 and 19, exponentially scale the radius from 2 pixels to 2 * (maxZoom - minZoom)^2 pixels.
            stop(19, Math.pow(2.0, 19 - 1.0) * 2f)
        )
    ),
    heatmapOpacity(0.75f)
)

```

The following video shows a map running the above code, which scales the radius while the map is being zoomed to create a consistent heat map rendering across zoom levels.



Next steps

For more code examples to add to your maps, see the following articles:

[Create a data source](#)

[Use data-driven style expressions](#)

Add an image layer to a map (Android SDK)

3/5/2021 • 3 minutes to read • [Edit Online](#)

This article shows you how to overlay an image to a fixed set of coordinates. Here are a few examples of different images types that can be overlaid on maps:

- Images captured from drones
- Building floorplans
- Historical or other specialized map images
- Blueprints of job sites

TIP

An image layer is an easy way to overlay an image on a map. Note that large images can consume a lot of memory and potentially cause performance issues. In this case, consider breaking your image up into tiles, and loading them into the map as a tile layer.

Add an image layer

The following code overlays an image of a [map of Newark, New Jersey, from 1922](#) on the map. This image is added to the `drawable` folder of the project. An image layer is created by setting the image and coordinates for the four corners in the format `[Top Left Corner, Top Right Corner, Bottom Right Corner, Bottom Left Corner]`. Adding image layers below the `label` layer is often desirable.

```
//Create an image layer.
ImageLayer layer = new ImageLayer(
    imageCoordinates(
        new Position[] {
            new Position(-74.22655, 40.773941), //Top Left Corner
            new Position(-74.12544, 40.773941), //Top Right Corner
            new Position(-74.12544, 40.712216), //Bottom Right Corner
            new Position(-74.22655, 40.712216) //Bottom Left Corner
        }
    ),
    setImage(R.drawable.newark_nj_1922)
);

//Add the image layer to the map, below the label layer.
map.layers.add(layer, "labels");
```

```

//Create an image layer.
val layer = ImageLayer(
    imageCoordinates(
        arrayOf<Position>(
            Position(-74.22655, 40.773941), //Top Left Corner
            Position(-74.12544, 40.773941), //Top Right Corner
            Position(-74.12544, 40.712216), //Bottom Right Corner
            Position(-74.22655, 40.712216) //Bottom Left Corner
        )
    ),
    setImage(R.drawable.newark_nj_1922)
)

//Add the image layer to the map, below the label layer.
map.layers.add(layer, "labels")

```

Alternatively, a URL to an image hosted on the online can be specified. However, if your scenario allows, add the image to your projects `drawable` folder, that will load faster since the image will be locally available and won't have to be downloaded.

```

//Create an image layer.
ImageLayer layer = new ImageLayer(
    imageCoordinates(
        new Position[] {
            new Position(-74.22655, 40.773941), //Top Left Corner
            new Position(-74.12544, 40.773941), //Top Right Corner
            new Position(-74.12544, 40.712216), //Bottom Right Corner
            new Position(-74.22655, 40.712216) //Bottom Left Corner
        }
    ),
    setUrl("https://www.lib.utexas.edu/maps/historical/newark_nj_1922.jpg")
);

//Add the image layer to the map, below the label layer.
map.layers.add(layer, "labels");

```

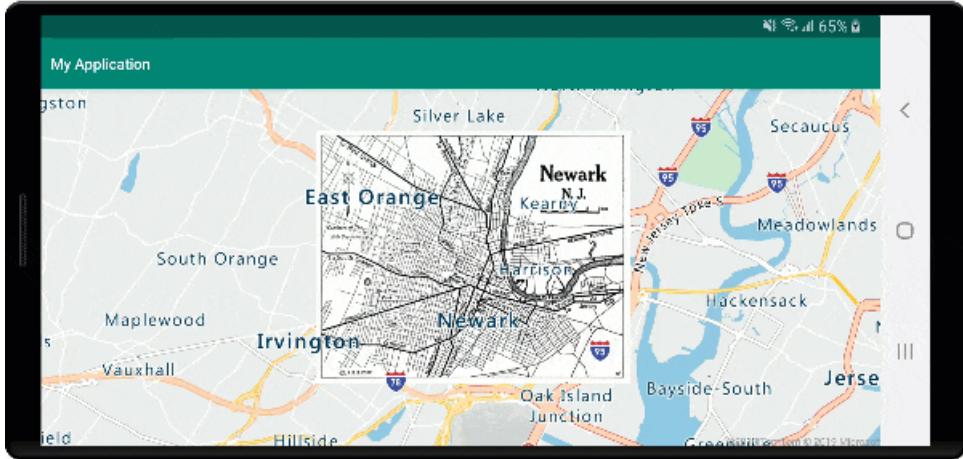
```

//Create an image layer.
val layer = ImageLayer(
    imageCoordinates(
        arrayOf<Position>(
            Position(-74.22655, 40.773941), //Top Left Corner
            Position(-74.12544, 40.773941), //Top Right Corner
            Position(-74.12544, 40.712216), //Bottom Right Corner
            Position(-74.22655, 40.712216) //Bottom Left Corner
        )
    ),
    setUrl("https://www.lib.utexas.edu/maps/historical/newark_nj_1922.jpg")
)

//Add the image layer to the map, below the label layer.
map.layers.add(layer, "labels")

```

The following screenshot shows a map of Newark, New Jersey, from 1922 overlaid using an image layer.



Import a KML file as ground overlay

This sample demonstrates how to add KML ground overlay information as an image layer on the map. KML ground overlays provide north, south, east, and west coordinates, and a counter-clockwise rotation. But, the image layer expects coordinates for each corner of the image. The KML ground overlay in this sample is for the Chartres cathedral, and it's sourced from [Wikimedia](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.2"
      xmlns:kml="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/2005/Atom">
  <GroundOverlay>
    <name>Map of Chartres cathedral</name>
    <Icon>
      <href>https://upload.wikimedia.org/wikipedia/commons/thumb/e/e3/Chartres.svg/1600px-Chartres.svg.png</href>
      <viewBoundScale>0.75</viewBoundScale>
    </Icon>
    <LatLonBox>
      <north>48.44820923628113</north>
      <south>48.44737203258976</south>
      <east>1.488833825534365</east>
      <west>1.486788581643038</west>
      <rotation>46.44067597839695</rotation>
    </LatLonBox>
  </GroundOverlay>
</kml>
```

The code uses the static `getCoordinatesFromEdges` method from the `ImageLayer` class. This method calculates the four corners of the image using the north, south, east, west, and rotation information of the KML ground overlay.

```

//Calculate the corner coordinates of the ground overlay.
Position[] corners = ImageLayer.getCoordinatesFromEdges(
    //North, south, east, west
    48.44820923628113, 48.44737203258976, 1.488833825534365, 1.486788581643038,

    //KML rotations are counter-clockwise, subtract from 360 to make them clockwise.
    360 - 46.44067597839695
);

//Create an image layer.
ImageLayer layer = new ImageLayer(
    imageCoordinates(corners),
    imageUrl("https://upload.wikimedia.org/wikipedia/commons/thumb/e/e3/Chartres.svg/1600px-Chartres.svg.png")
);

//Add the image layer to the map, below the label layer.
map.layers.add(layer, "labels");

```

```

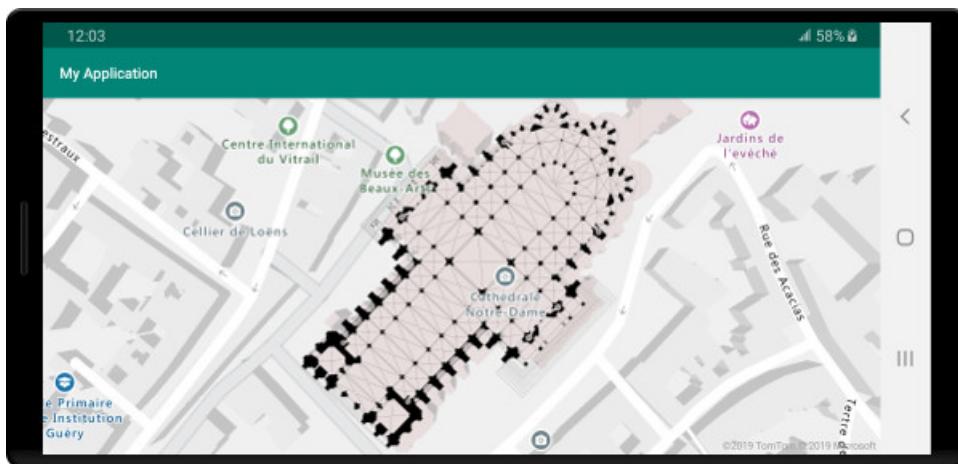
//Calculate the corner coordinates of the ground overlay.
val corners: Array<Position> =
    ImageLayer.getCoordinatesFromEdges( //North, south, east, west
        48.44820923628113,
        48.44737203258976,
        1.488833825534365,
        1.486788581643038, //KML rotations are counter-clockwise, subtract from 360 to make them clockwise.
        360 - 46.44067597839695
    )

//Create an image layer.
val layer = ImageLayer(
    imageCoordinates(corners),
    imageUrl("https://upload.wikimedia.org/wikipedia/commons/thumb/e/e3/Chartres.svg/1600px-Chartres.svg.png")
)

//Add the image layer to the map, below the label layer.
map.layers.add(layer, "labels")

```

The following screenshot shows a map with a KML ground overlay overlaid using an image layer.



TIP

Use the `getPixels` and `getPositions` methods of the image layer class to convert between geographic coordinates of the positioned image layer and the local image pixel coordinates.

Next steps

See the following article to learn more about ways to overlay imagery on a map.

[Add a tile layer](#)

Add a tile layer to a map (Android SDK)

3/26/2021 • 4 minutes to read • [Edit Online](#)

This article shows you how to render a tile layer on a map using the Azure Maps Android SDK. Tile layers allow you to superimpose images on top of Azure Maps base map tiles. More information on Azure Maps tiling system can be found in the [Zoom levels and tile grid](#) documentation.

A Tile layer loads in tiles from a server. These images can be pre-rendered and stored like any other image on a server, using a naming convention that the tile layer understands. Or, these images can be rendered with a dynamic service that generates the images near real time. There are three different tile service naming conventions supported by Azure Maps TileLayer class:

- X, Y, Zoom notation - Based on the zoom level, x is the column and y is the row position of the tile in the tile grid.
- Quadkey notation - Combination x, y, zoom information into a single string value that is a unique identifier for a tile.
- Bounding Box - Bounding box coordinates can be used to specify an image in the format `{west},{south},{east},{north}`, which is commonly used by [web-mapping Services \(WMS\)](#).

TIP

A TileLayer is a great way to visualize large data sets on the map. Not only can a tile layer be generated from an image, but vector data can also be rendered as a tile layer too. By rendering vector data as a tile layer, the map control only needs to load the tiles, which can be much smaller in file size than the vector data they represent. This technique is used by many who need to render millions of rows of data on the map.

The tile URL passed into a Tile layer must be an http/https URL to a TileJSON resource or a tile URL template that uses the following parameters:

- `{x}` - X position of the tile. Also needs `{y}` and `{z}`.
- `{y}` - Y position of the tile. Also needs `{x}` and `{z}`.
- `{z}` - Zoom level of the tile. Also needs `{x}` and `{y}`.
- `{quadkey}` - Tile quadkey identifier based on the Bing Maps tile system naming convention.
- `{bbox-epsg-3857}` - A bounding box string with the format `{west},{south},{east},{north}` in the EPSG 3857 Spatial Reference System.
- `{subdomain}` - A placeholder for the subdomain values, if the subdomain value is specified.
- `azmapsdomain.invalid` - A placeholder to align the domain and authentication of tile requests with the same values used by the map. Use this when calling a tile service hosted by Azure Maps.

Prerequisites

To complete the process in this article, you need to install [Azure Maps Android SDK](#) to load a map.

Add a tile layer to the map

This sample shows how to create a tile layer that points to a set of tiles. This sample uses the "x, y, zoom" tiling system. The source of this tile layer is the [OpenSeaMap project](#), which contains crowd sourced nautical charts. Often when viewing tile layers it is desirable to be able to clearly see the labels of cities on the map. This behavior can be achieved by inserting the tile layer below the map label layers.

```

TileLayer layer = new TileLayer(
    tileUrl("https://tiles.openseamap.org/seamark/{z}/{x}/{y}.png"),
    opacity(0.8f),
    tileSize(256),
    minSourceZoom(7),
    maxSourceZoom(17)
);

map.layers.add(layer, "labels");

```

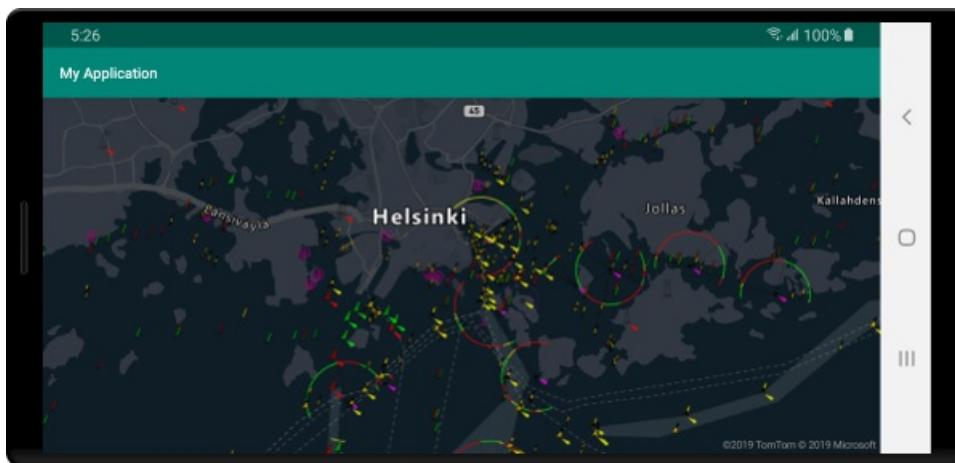
```

val layer = TileLayer(
    tileUrl("https://tiles.openseamap.org/seamark/{z}/{x}/{y}.png"),
    opacity(0.8f),
    tileSize(256),
    minSourceZoom(7),
    maxSourceZoom(17)
)

map.layers.add(layer, "labels")

```

The following screenshot shows the above code displaying a tile layer of nautical information on a map that has a dark grayscale style.



Add an OGC web-mapping service (WMS)

A web-mapping service (WMS) is an Open Geospatial Consortium (OGC) standard for serving images of map data. There are many open data sets available in this format that you can use with Azure Maps. This type of service can be used with a tile layer if the service supports the `EPSG:3857` coordinate reference system (CRS). When using a WMS service, set the width and height parameters to the same value that is supported by the service, be sure to set this same value in the `tileSize` option. In the formatted URL, set the `BBOX` parameter of the service with the `{bbox-epsg-3857}` placeholder.

```

TileLayer layer = new TileLayer(
    tileUrl("https://mrdata.usgs.gov/services/gscworld?
FORMAT=image/png&HEIGHT=1024&LAYERS=geology&REQUEST=GetMap&STYLES=default&TILED=true&TRANSPARENT=true&WIDTH=
1024&VERSION=1.3.0&SERVICE=WMS&CRS=EPSG:3857&BBOX={bbox-epsg-3857}"),
    tileSize(1024)
);

map.layers.add(layer, "labels");

```

```

val layer = TileLayer(
    tileUrl("https://mrdata.usgs.gov/services/gscworld?
FORMAT=image/png&HEIGHT=1024&LAYERS=geology&REQUEST=GetMap&STYLES=default&TILED=true&TRANSPARENT=true&WIDTH=
1024&VERSION=1.3.0&SERVICE=WMS&CRS=EPSG:3857&BBOX={bbox-epsg-3857}"),
    tileSize(1024)
)

map.layers.add(layer, "labels")

```

The following screenshot shows the above code overlaying a web-mapping service of geological data from the [U.S. Geological Survey \(USGS\)](#) on top of a map, below the labels.



Add an OGC web-mapping tile service (WMTS)

A web-mapping tile service (WMTS) is an Open Geospatial Consortium (OGC) standard for serving tiled based overlays for maps. There are many open data sets available in this format that you can use with Azure Maps.

This type of service can be used with a tile layer if the service supports the `EPSG:3857` or `GoogleMapsCompatible` coordinate reference system (CRS). When using a WMTS service, set the width and height parameters to the same value that is supported by the service, be sure to set this same value in the `tileSize` option. In the formatted URL, replace the following placeholders accordingly:

- `{TileMatrix}` => `{z}`
- `{TileRow}` => `{y}`
- `{TileCol}` => `{x}`

```

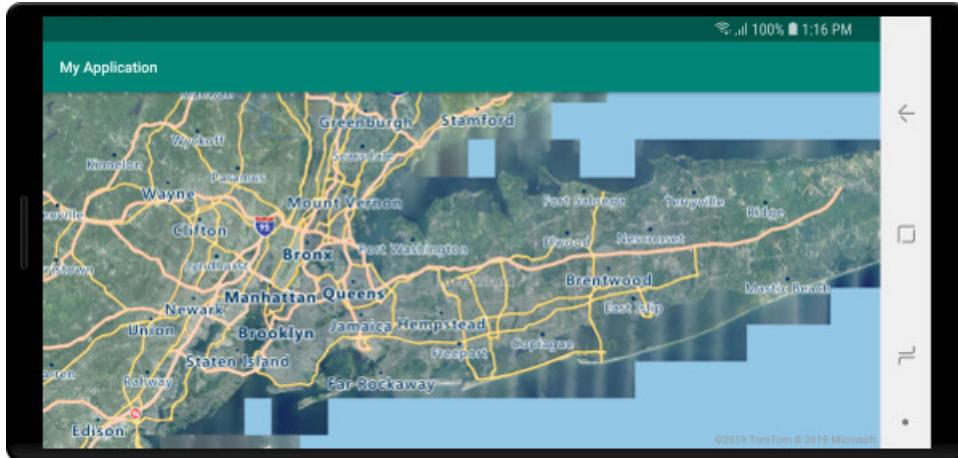
TileLayer layer = new TileLayer(
    tileUrl("https://basemap.nationalmap.gov/arcgis/rest/services/USGSImageryOnly/MapServer/WMTS/tile/1.0.0/USGS
    ImageryOnly/default/GoogleMapsCompatible/{z}/{y}/{x}"),
    tileSize(256),
    bounds(-173.25000107492872, 0.0005794121990209753, 146.12527718104752, 71.506811402077),
    maxSourceZoom(18)
);

map.layers.add(layer, "transit");

```

```
val layer = TileLayer(  
  
    tileUrl("https://basemap.nationalmap.gov/arcgis/rest/services/USGSImageryOnly/MapServer/WMTS/tile/1.0.0/USGS  
    ImageryOnly/default/GoogleMapsCompatible/{z}/{y}/{x}"),  
    tileSize(256),  
    bounds(-173.25000107492872, 0.0005794121990209753, 146.12527718104752, 71.506811402077),  
    maxSourceZoom(18)  
)  
  
map.layers.add(layer, "transit")
```

The following screenshot shows the above code overlaying a web-mapping tile service of imagery from the [U.S. Geological Survey \(USGS\) National Map](#) on top of a map, below the roads and labels.



Next steps

See the following article to learn more about ways to overlay imagery on a map.

[Image layer](#)

Show traffic data on the map (Android SDK)

3/26/2021 • 3 minutes to read • [Edit Online](#)

Flow data and incidents data are the two types of traffic data that can be displayed on the map. This guide shows you how to display both types of traffic data. Incidents data consists of point and line-based data for things such as constructions, road closures, and accidents. Flow data shows metrics about the flow of traffic on the road.

Prerequisites

Be sure to complete the steps in the [Quickstart: Create an Android app](#) document. Code blocks in this article can be inserted into the maps `onReady` event handler.

Show traffic on the map

There are two types of traffic data available in Azure Maps:

- Incident data - consists of point and line-based data for things such as construction, road closures, and accidents.
- Flow data - provides metrics on the flow of traffic on the roads. Often, traffic flow data is used to color the roads. The colors are based on how much traffic is slowing down the flow, relative to the speed limit, or another metric. There are four values that can be passed into the traffic `flow` option of the map.

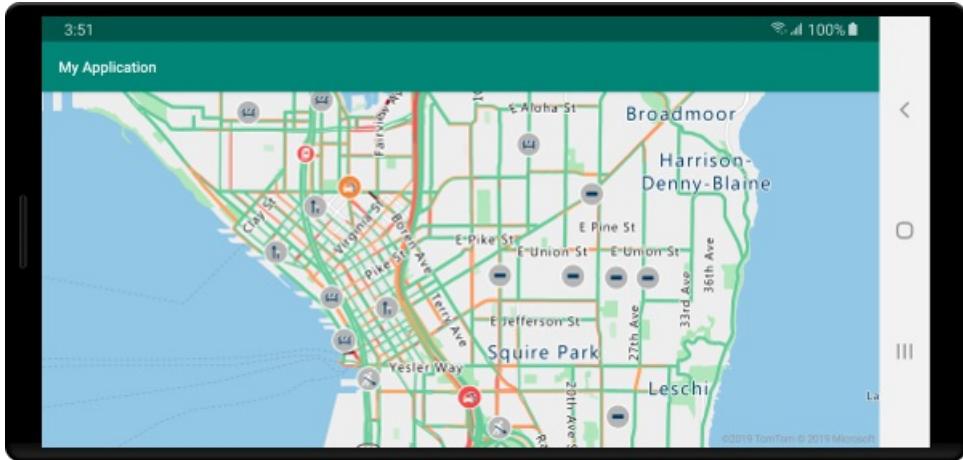
FLOW VALUE	DESCRIPTION
TrafficFlow.NONE	Doesn't display traffic data on the map
TrafficFlow.RELATIVE	Shows traffic data that's relative to the free-flow speed of the road
TrafficFlow.RELATIVE_DELAY	Displays areas that are slower than the average expected delay
TrafficFlow.ABSOLUTE	Shows the absolute speed of all vehicles on the road

The following code shows how to display traffic data on the map.

```
//Show traffic on the map using the traffic options.  
map.setTraffic(  
    incidents(true),  
    flow(TrafficFlow.RELATIVE)  
)
```

```
map.setTraffic(  
    incidents(true),  
    flow(TrafficFlow.RELATIVE)  
)
```

The following screenshot shows the above code rendering real-time traffic information on the map.



Get traffic incident details

Details about a traffic incident are available within the properties of the feature used to display the incident on the map. Traffic incidents are added to the map using the Azure Maps traffic incident vector tile service. The format of the data in these vector tiles is [documented here](#). The following code adds a click event to the map and retrieves the traffic incident feature that was clicked and displays a toast message with some of the details.

```

//Show traffic information on the map.
map.setTraffic(
    incidents(true),
    flow(TrafficFlow.RELATIVE)
);

//Add a click event to the map.
map.events.add((OnFeatureClick) (features) -> {

    if (features != null && features.size() > 0) {
        Feature incident = features.get(0);

        //Ensure that the clicked feature is an traffic incident feature.
        if (incident.properties() != null && incident.hasProperty("incidentType")) {

            StringBuilder sb = new StringBuilder();
            String incidentType = incident.getStringProperty("incidentType");

            if (incidentType != null) {
                sb.append(incidentType);
            }

            if (sb.length() > 0) {
                sb.append("\n");
            }

            //If the road is closed, find out where it is closed from.
            if ("Road Closed".equals(incidentType)) {
                String from = incident.getStringProperty("from");

                if (from != null) {
                    sb.append(from);
                }
            } else {
                //Get the description of the traffic incident.
                String description = incident.getStringProperty("description");

                if (description != null) {
                    sb.append(description);
                }
            }

            String message = sb.toString();

            if (message.length() > 0) {
                Toast.makeText(this, message, Toast.LENGTH_LONG).show();
            }
        }
    }
});
```

```

//Show traffic information on the map.
map.setTraffic(
    incidents(true),
    flow(TrafficFlow.RELATIVE)
)

//Add a click event to the map.
map.events.add(OnFeatureClick { features: List<Feature>? ->
    if (features != null && features.size > 0) {
        val incident = features[0]

        //Ensure that the clicked feature is an traffic incident feature.
        if (incident.properties() != null && incident.hasProperty("incidentType")) {
            val sb = StringBuilder()
            val incidentType = incident.getStringProperty("incidentType")

            if (incidentType != null) {
                sb.append(incidentType)
            }

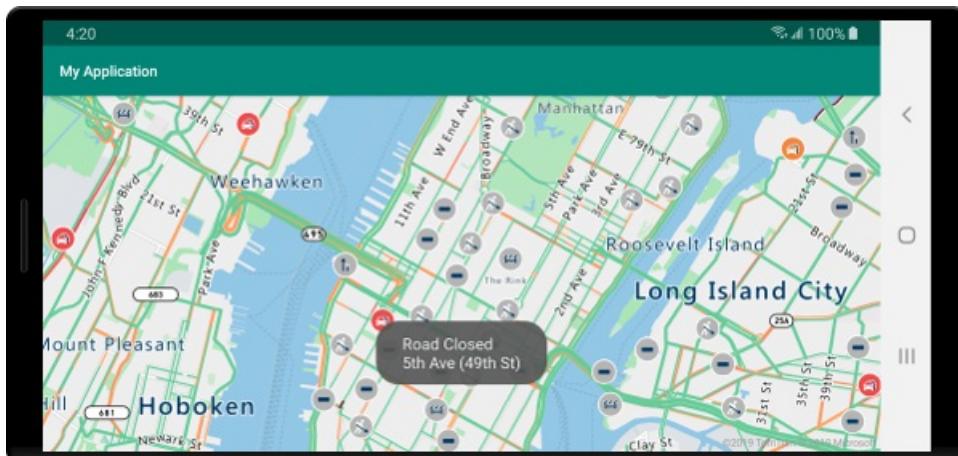
            if (sb.length > 0) {
                sb.append("\n")
            }
        }

        //If the road is closed, find out where it is closed from.
        if ("Road Closed" == incidentType) {
            val from = incident.getStringProperty("from")
            if (from != null) {
                sb.append(from)
            }
        } else { //Get the description of the traffic incident.
            val description = incident.getStringProperty("description")
            if (description != null) {
                sb.append(description)
            }
        }
    }

    val message = sb.toString()
    if (message.length > 0) {
        Toast.makeText(this, message, Toast.LENGTH_LONG).show()
    }
}
)
})

```

The following screenshot shows the above code rendering real-time traffic information on the map with a toast message displaying incident details.



Next steps

View the following guides to learn how to add more data to your map:

[Add a tile layer](#)

Data-driven style expressions (Android SDK)

3/5/2021 • 28 minutes to read • [Edit Online](#)

Expressions enable you to apply business logic to styling options that observe the properties defined in each shape in a data source. Expressions can filter data in a data source or a layer. Expressions may consist of conditional logic, like if-statements. And, they can be used to manipulate data using: string operators, logical operators, and mathematical operators.

Data-driven styles reduce the amount of code needed to implement business logic around styling. When used with layers, expressions are evaluated at render time on a separate thread. This functionality provides increased performance compared to evaluating business logic on the UI thread.

The Azure Maps Android SDK supports nearly all the same style expressions as the Azure Maps Web SDK, so all the same concepts outlined in the [Data-driven Style Expressions \(Web SDK\)](#) can be carried over into an Android app. All style expressions in the Azure Maps Android SDK are available under the

`com.microsoft.azure.maps.mapcontrol.options.Expression` namespace. There are many different types of style expressions.

Type of Expressions	Description
Boolean expressions	Boolean expressions provide a set of boolean operators expressions for evaluating boolean comparisons.
Color expressions	Color expressions make it easier to create and manipulate color values.
Conditional expressions	Conditional expressions provide logic operations that are like if-statements.
Data expressions	Provides access to the property data in a feature.
Interpolate and Step expressions	Interpolate and step expressions can be used to calculate values along an interpolated curve or step function.
JSON-based expressions	Makes it easy to reuse style raw JSON-based expressions created for the Web SDK with the Android SDK.
Layer specific expressions	Special expressions that are only applicable to a single layer.
Math expressions	Provides mathematical operators to perform data-driven calculations within the expression framework.
String operator expressions	String operator expressions perform conversion operations on strings such as concatenating and converting the case.
Type expressions	Type expressions provide tools for testing and converting different data types like strings, numbers, and boolean values.

TYPE OF EXPRESSIONS	DESCRIPTION
Variable binding expressions	Variable binding expressions store the results of a calculation in a variable and referenced elsewhere in an expression multiple times without having to recalculate the stored value.
Zoom expression	Retrieves the current zoom level of the map at render time.

NOTE

The syntax for expressions is largely identical in Java and Kotlin. If you have the documentation set to Kotlin, but see code blocks for Java, the code is identical in both languages.

All examples in this section of the document use the following feature to demonstrate different ways in which these expressions can be used.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [-122.13284, 47.63699]
  },
  "properties": {
    "id": 123,
    "entityType": "restaurant",
    "revenue": 12345,
    "subTitle": "Building 40",
    "temperature": 64,
    "title": "Cafeteria",
    "zoneColor": "purple",
    "abcArray": ["a", "b", "c"],
    "array2d": [["a", "b"], ["x", "y"]],
    "_style": {
      "fillColor": "red"
    }
  }
}
```

The following code shows how to manually create this GeoJSON feature in an app.

```
//Create a point feature.
Feature feature = Feature.fromGeometry(Point.fromLngLat(-100, 45));

//Add properties to the feature.
feature.addNumberProperty("id", 123);
feature.addStringProperty("entityType", "restaurant");
feature.addNumberProperty("revenue", 12345);
feature.addStringProperty("subTitle", "Building 40");
feature.addNumberProperty("temperature", 64);
feature.addStringProperty("title", "Cafeteria");
feature.addStringProperty("zoneColor", "purple");

JSONArray abcArray = new JSONArray();
abcArray.add("a");
abcArray.add("b");
abcArray.add("c");

feature.addProperty("abcArray", abcArray);

JSONArray array1 = new JSONArray();
array1.add("a");
array1.add("b");

JSONArray array2 = new JSONArray();
array1.add("x");
array1.add("y");

JSONArray array2d = new JSONArray();
array2d.add(array1);
array2d.add(array2);

feature.addProperty("array2d", array2d);

JsonObject style = new JsonObject();
style.addProperty("fillColor", "red");

feature.addProperty("_style", style);
```

```

//Create a point feature.
val feature = Feature.fromGeometry(Point.fromLngLat(-100, 45))

//Add properties to the feature.
feature.addNumberProperty("id", 123)
feature.addStringProperty("entityType", "restaurant")
feature.addNumberProperty("revenue", 12345)
feature.addStringProperty("subTitle", "Building 40")
feature.addNumberProperty("temperature", 64)
feature.addStringProperty("title", "Cafeteria")
feature.addStringProperty("zoneColor", "purple")

val abcArray = JSONArray()
abcArray.add("a")
abcArray.add("b")
abcArray.add("c")

feature.addProperty("abcArray", abcArray)

val array1 = JSONArray()
array1.add("a")
array1.add("b")

val array2 = JSONArray()
array2.add("x")
array2.add("y")

val array2d = JSONArray()
array2d.add(array1)
array2d.add(array2)

feature.addProperty("array2d", array2d)

val style = JSONObject()
style.addProperty("fillColor", "red")

feature.addProperty("_style", style)

```

The following code shows how to deserialize the stringified version of the JSON object into a GeoJSON feature in an app.

```

String featureString = "{\"type\":\"Feature\",\"geometry\":{\"type\":\"Point\",\"coordinates\":[-122.13284,47.63699]},\"properties\":{\\"id\\":123,\"entityType\":\"restaurant\",\"revenue\":12345,\"subTitle\":\"Building 40\",\"temperature\":64,\"title\":\"Cafeteria\",\"zoneColor\":\"purple\",\"abcArray\":[\"a\",\"b\",\"c\"],\"array2d\":[[\"a\",\"b\"],[\"x\",\"y\"]],\"_style\":{\"fillColor\":\"red\"}}}";

Feature feature = Feature.fromJson(featureString);

```

```

val featureString = "{\"type\":\"Feature\",\"geometry\":{\"type\":\"Point\",\"coordinates\":[-122.13284,47.63699]},\"properties\":{\\"id\\":123,\"entityType\":\"restaurant\",\"revenue\":12345,\"subTitle\":\"Building 40\",\"temperature\":64,\"title\":\"Cafeteria\",\"zoneColor\":\"purple\",\"abcArray\":[\"a\",\"b\",\"c\"],\"array2d\":[[\"a\",\"b\"],[\"x\",\"y\"]],\"_style\":{\"fillColor\":\"red\"}}}"

val feature = Feature.fromJson(featureString)

```

JSON-based expressions

The Azure Maps Web SDK also supports data-driven style expressions that are represented using a JSON array. These same expressions can be recreated using the native `Expression` class in the Android SDK. Alternatively,

these JSON-based expressions can be converted into a string using a web function such as `JSON.stringify` and passed into the `Expression.raw(String rawExpression)` method. For example, take the following JSON expression.

```
var exp = ['get','title'];
JSON.stringify(exp); // = "[['get','title']]"
```

The stringified version of the above expression would be `"[['get','title']]"` and can be read into the Android SDK as follows.

```
Expression exp = Expression.raw("[['get','title']]")
```

```
val exp = Expression.raw("[['get','title']]")
```

Using this approach can make it easy to reuse style expressions between mobile and web apps that use Azure Maps.

This video provides an overview of data-driven styling in Azure Maps.

Data expressions

Data expressions provide access to the property data in a feature.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>accumulated()</code>	number	Gets the value of a cluster property accumulated so far.
<code>at(number Expression, Expression)</code>	value	Retrieves an item from an array.
<code>geometryType()</code>	string	Gets the feature's geometry type: Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon.
<code>get(string Expression) get(string Expression, Expression)</code>	value	Gets the property value from the properties of the provided object. Returns null if the requested property is missing.
<code>has(string Expression) has(string Expression, Expression)</code>	boolean	Determines if the properties of a feature have the specified property.
<code>id()</code>	value	Gets the feature's ID if it has one.
<code>in(string number Expression, Expression)</code>	boolean	Determines whether an item exists in an array
<code>length(string Expression)</code>	number	Gets the length of a string or an array.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>properties()</code>	value	Gets the feature properties object.

The following Web SDK style expressions are not supported in the Android SDK:

- index-of
- slice

Examples

Properties of a feature can be accessed directly in an expression by using a `get` expression. This example uses the `zoneColor` value of the feature to specify the color property of a bubble layer.

```
BubbleLayer layer = new BubbleLayer(source,
    //Get the zoneColor value.
    bubbleColor(get("zoneColor"))
);
```

```
val layer = BubbleLayer(source,
    //Get the zoneColor value.
    bubbleColor(get("zoneColor"))
)
```

The above example will work fine, if all the point features have the `zoneColor` property. If they don't, the color will likely fall back to "black". To modify the fallback color, use a `switchCase` expression in combination with the `has` expression to check if the property exists. If the property doesn't exist, return a fallback color.

```
BubbleLayer layer = new BubbleLayer(source,
    bubbleColor(
        //Use a conditional case expression.
        switchCase(
            //Check to see if feature has a "zoneColor"
            has("zoneColor"),
            //If it does, use it.
            get("zoneColor"),
            //If it doesn't, default to blue.
            literal("blue")
        )
    );
);
```

```

val layer = BubbleLayer(source,
    bubbleColor(
        //Use a conditional case expression.
        switchCase(
            //Check to see if feature has a "zoneColor"
            has("zoneColor"),
                //If it does, use it.
                get("zoneColor"),
                //If it doesn't, default to blue.
                literal("blue")
        )
    )
)

```

Bubble and symbol layers will render the coordinates of all shapes in a data source, by default. This behavior can highlight the vertices of a polygon or a line. The `filter` option of the layer can be used to limit the geometry type of the features it renders, by using a `geometryType` expression within a boolean expression. The following example limits a bubble layer so that only `Point` features are rendered.

```

BubbleLayer layer = new BubbleLayer(source,
    filter(eq(geometryType(), "Point"))
);

```

```

val layer = BubbleLayer(source,
    filter(eq(geometryType(), "Point"))
)

```

The following example allows both `Point` and `MultiPoint` features to be rendered.

```

BubbleLayer layer = new BubbleLayer(source,
    filter(any(eq(geometryType(), "Point"), eq(geometryType(), "MultiPoint")))
);

```

```

val layer = BubbleLayer(source,
    filter(any(eq(geometryType(), "Point"), eq(geometryType(), "MultiPoint")))
)

```

Similarly, the outline of Polygons will render in line layers. To disable this behavior in a line layer, add a filter that only allows `LineString` and `MultiLineString` features.

Here are some additional examples of how to use data expressions:

```

//Get item [2] from an array "properties.abcArray[1]" = "c"
at(2, get("abcArray"))

//Get item [0][1] from a 2D array "properties.array2d[0][1]" = "b"
at(1, at(0, get("array2d")))

//Check to see if a value is in an array "properties.abcArray.indexOf('a') !== -1" = true
in("a", get("abcArray"))

//Get the length of an array "properties.abcArray.length" = 3
length(get("abcArray"))

//Get the value of a subproperty "properties._style.fillColor" = "red"
get("fillColor", get("_style"))

//Check that "fillColor" exists as a subproperty of "_style".
has("fillColor", get("_style"))

```

Math expressions

Math expressions provide mathematical operators to perform data-driven calculations within the expression framework.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>abs(number Expression)</code>	number	Calculates the absolute value of the specified number.
<code>acos(number Expression)</code>	number	Calculates the arccosine of the specified number.
<code>asin(number Expression)</code>	number	Calculates the arcsine of the specified number.
<code>atan(number Expression)</code>	number	Calculates the arctangent of the specified number.
<code>ceil(number Expression)</code>	number	Rounds the number up to the next whole integer.
<code>cos(number Expression)</code>	number	Calculates the cos of the specified number.
<code>division(number, number) division(Expression, Expression)</code>	number	Divides the first number by the second number. Web SDK equivalent expression: <code>/</code>
<code>e()</code>	number	Returns the mathematical constant <code>e</code> .
<code>floor(number Expression)</code>	number	Rounds the number down to the previous whole integer.
<code>log10(number Expression)</code>	number	Calculates the base-ten logarithm of the specified number.
<code>log2(number Expression)</code>	number	Calculates the base-two logarithm of the specified number.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>ln(number Expression)</code>	number	Calculates the natural logarithm of the specified number.
<code>ln2()</code>	number	Returns the mathematical constant <code>ln(2)</code> .
<code>max(numbers... expressions...)</code>	number	Calculates the maximum number in the specified set of numbers.
<code>min(numbers... expressions...)</code>	number	Calculates the minimum number in the specified set of numbers.
<code>mod(number, number) mod(Expression, Expression)</code>	number	Calculates the remainder when dividing the first number by the second number. Web SDK equivalent expression: <code>%</code>
<code>pi()</code>	number	Returns the mathematical constant <code>PI</code> .
<code>pow(number, number) pow(Expression, Expression)</code>	number	Calculates the value of the first value raised to the power of the second number.
<code>product(numbers... expressions...)</code>	number	Multiplies the specified numbers together. Web SDK equivalent expression: <code>*</code>
<code>round(number Expression)</code>	number	Rounds the number to the nearest integer. Halfway values are rounded away from zero. For example, <code>round(-1.5)</code> evaluates to <code>-2</code> .
<code>sin(number Expression)</code>	number	Calculates the sine of the specified number.
<code>sqrt(number Expression)</code>	number	Calculates the square root of the specified number.
<code>subtract(number Expression)</code>	number	Subtracts 0 by the specified number.
<code>subtract(number Expression, number Expression)</code>	number	Subtracts the first numbers by the second number.
<code>sum(numbers... expressions...)</code>	number	Calculates the sum of the specified numbers.
<code>tan(number Expression)</code>	number	Calculates the tangent of the specified number.

Boolean expressions

Boolean expressions provide a set of boolean operators expressions for evaluating boolean comparisons.

When comparing values, the comparison is strictly typed. Values of different types are always considered unequal. Cases where the types are known to be different at parse time are considered invalid and will produce a parse error.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>all(Expression...)</code>	boolean	Returns <code>true</code> if all the inputs are <code>true</code> , <code>false</code> otherwise.
<code>any(Expression...)</code>	boolean	Returns <code>true</code> if any of the inputs are <code>true</code> , <code>false</code> otherwise.
<pre>eq(Expression compareOne, Expression boolean number string compareTwo) eq(Expression compareOne, Expression string compareTwo, Expression collator)</pre>	boolean	Returns <code>true</code> if the input values are equal, <code>false</code> otherwise. The arguments are required to be either both strings or both numbers.
<pre>gt(Expression compareOne, Expression boolean number string compareTwo) gt(Expression compareOne, Expression string compareTwo, Expression collator)</pre>	boolean	Returns <code>true</code> if the first input is strictly greater than the second, <code>false</code> otherwise. The arguments are required to be either both strings or both numbers.
<pre>gte(Expression compareOne, Expression boolean number string compareTwo) gte(Expression compareOne, Expression string compareTwo, Expression collator)</pre>	boolean	Returns <code>true</code> if the first input is greater than or equal to the second, <code>false</code> otherwise. The arguments are required to be either both strings or both numbers.
<pre>lt(Expression compareOne, Expression boolean number string compareTwo) lt(Expression compareOne, Expression string compareTwo, Expression collator)</pre>	boolean	Returns <code>true</code> if the first input is strictly less than the second, <code>false</code> otherwise. The arguments are required to be either both strings or both numbers.
<pre>lte(Expression compareOne, Expression boolean number string compareTwo) lte(Expression compareOne, Expression string compareTwo, Expression collator)</pre>	boolean	Returns <code>true</code> if the first input is less than or equal to the second, <code>false</code> otherwise. The arguments are required to be either both strings or both numbers.
<pre>neq(Expression compareOne, Expression boolean number string compareTwo) neq(Expression compareOne, Expression string compareTwo, Expression collator)</pre>	boolean	Returns <code>true</code> if the input values are not equal, <code>false</code> otherwise.
<code>not(Expression boolean)</code>	boolean	Logical negation. Returns <code>true</code> if the input is <code>false</code> , and <code>false</code> if the input is <code>true</code> .

Conditional expressions

Conditional expressions provide logic operations that are like if-statements.

The following expressions perform conditional logic operations on the input data. For example, the `switchCase` expression provides "if/then/else" logic while the `match` expression is like a "switch-statement".

Switch case expression

A `switchCase` expression is a type of conditional expression that provides "if/then/else" logic. This type of expression steps through a list of boolean conditions. It returns the output value of the first boolean condition to evaluate to true.

The following pseudocode defines the structure of the `switchCase` expression.

```
switchCase(  
    condition1: boolean expression,  
    output1: value,  
    condition2: boolean expression,  
    output2: value,  
    ...  
    fallback: value  
)
```

Example

The following example steps through different boolean conditions until it finds one that evaluates to `true`, and then returns that associated value. If no boolean condition evaluates to `true`, a fallback value will be returned.

```
BubbleLayer layer = new BubbleLayer(source,  
    bubbleColor(  
        switchCase(  
            //Check to see if the first boolean expression is true, and if it is, return its assigned  
            result.  
            //If it has a zoneColor property, use its value as a color.  
            has("zoneColor"), toColor(get("zoneColor")),  
  
            //Check to see if the second boolean expression is true, and if it is, return its assigned  
            result.  
            //If it has a temperature property with a value greater than or equal to 100, make it red.  
            all(has("temperature"), gte(get("temperature"), 100)), color(Color.RED),  
  
            //Specify a default value to return. In this case green.  
            color(Color.GREEN)  
        )  
    )  
);
```

```

val layer = BubbleLayer(source,
    bubbleColor(
        switchCase(
            //Check to see if the first boolean expression is true, and if it is, return its assigned
            result.
            //If it has a zoneColor property, use its value as a color.
            has("zoneColor"), toColor(get("zoneColor")),

            //Check to see if the second boolean expression is true, and if it is, return its assigned
            result.
            //If it has a temperature property with a value greater than or equal to 100, make it red.
            all(has("temperature"), gte(get("temperature"), 100)), color(Color.RED),

            //Specify a default value to return. In this case green.
            color(Color.GREEN)
        )
    )
)

```

Match expression

A `match` expression is a type of conditional expression that provides switch-statement like logic. The input can be any expression such as `get("entityType")` that returns a string or a number. Each stop must have a label that is either a single literal value or an array of literal values, whose values must be all strings or all numbers. The input matches if any of the values in the array match. Each stop label must be unique. If the input type doesn't match the type of the labels, the result will be the default fallback value.

The following pseudocode defines the structure of the `match` expression.

```
match(Expression input, Expression defaultOutput, Expression.Stop... stops)
```

Examples

The following example looks at the `entityType` property of a Point feature in a bubble layer searches for a match. If it finds a match, that specified value is returned or it returns the fallback value.

```

BubbleLayer layer = new BubbleLayer(source,
    bubbleColor(
        match(
            //Get the input value to match.
            get("entityType"),

            //Specify a default value to return if no match is found.
            color(Color.BLACK),

            //List the values to match and the result to return for each match.

            //If value is "restaurant" return "red".
            stop("restaurant", color(Color.RED)),

            //If value is "park" return "green".
            stop("park", color(Color.GREEN))
        )
    );
)
```

```

val layer = BubbleLayer(source,
    bubbleColor(
        match(
            //Get the input value to match.
            get("entityType"),

            //Specify a default value to return if no match is found.
            color(Color.BLACK),

            //List the values to match and the result to return for each match.

            //If value is "restaurant" return "red".
            stop("restaurant", color(Color.RED)),

            //If value is "park" return "green".
            stop("park", color(Color.GREEN))
        )
    )
)

```

The following example uses an array to list a set of labels that should all return the same value. This approach is much more efficient than listing each label individually. In this case, if the `entityType` property is "restaurant" or "grocery_store", the color "red" will be returned.

```

BubbleLayer layer = new BubbleLayer(source,
    bubbleColor(
        match(
            //Get the input value to match.
            get("entityType"),

            //Specify a default value to return if no match is found.
            color(Color.BLACK),

            //List the values to match and the result to return for each match.

            //If value is "restaurant" or "grocery_store" return "red".
            stop(Arrays.asList("restaurant", "grocery_store"), color(Color.RED)),

            //If value is "park" return "green".
            stop("park", color(Color.GREEN))
        )
    );
)
```

```

val layer = BubbleLayer(source,
    bubbleColor(
        match(
            //Get the input value to match.
            get("entityType"),

            //Specify a default value to return if no match is found.
            color(Color.BLACK),

            //List the values to match and the result to return for each match.

            //If value is "restaurant" or "grocery_store" return "red".
            stop(arrayOf("restaurant", "grocery_store"), color(Color.RED)),

            //If value is "park" return "green".
            stop("park", color(Color.GREEN))
        )
    )
)

```

Coalesce expression

A `coalesce` expression steps through a set of expressions until the first non-null value is obtained and returns that value.

The following pseudocode defines the structure of the `coalesce` expression.

```
coalesce(Expression... input)
```

Example

The following example uses a `coalesce` expression to set the `textField` option of a symbol layer. If the `title` property is missing from the feature or set to `null`, the expression will then try looking for the `subTitle` property, if its missing or `null`, it will then fall back to an empty string.

```

SymbolLayer layer = new SymbolLayer(source,
    textField(
        coalesce(
            //Try getting the title property.
            get("title"),

            //If there is no title, try getting the subTitle.
            get("subTitle"),

            //Default to an empty string.
            literal("")
        )
    );

```

```

val layer = SymbolLayer(source,
    textField(
        coalesce(
            //Try getting the title property.
            get("title"),

            //If there is no title, try getting the subTitle.
            get("subTitle"),

            //Default to an empty string.
            literal("")
        )
    )
)

```

Type expressions

Type expressions provide tools for testing and converting different data types like strings, numbers, and boolean values.

EXPRESSION	RETURN TYPE	DESCRIPTION
array(Expression)	Object[]	Asserts that the input is an array.
bool(Expression)	boolean	Asserts that the input value is a boolean.
collator(boolean caseSensitive, boolean diacriticSensitive) collator(boolean caseSensitive, boolean diacriticSensitive, java.util.Locale locale) collator(Expression caseSensitive, Expression diacriticSensitive) collator(Expression caseSensitive, Expression diacriticSensitive, Expression locale)	collator	Returns a collator for use in locale-dependent comparison operations. The case-sensitive and diacritic-sensitive options default to false. The locale argument specifies the IETF language tag of the locale to use. If none is provided, the default locale is used. If the requested locale is not available, the collator will use a system-defined fallback locale. Use resolved-locale to test the results of locale fallback behavior.
literal(boolean number string Object Object[])	boolean number string Object Object[]	Returns a literal array or object value. Use this expression to prevent an array or object from being evaluated as an expression. This is necessary when an array or object needs to be returned by an expression.
number(Expression)	number	Asserts that the input value is a number.
object(Expression)	Object	Asserts that the input value is an object.
string(Expression)	string	Asserts that the input value is a string.
toArray(Expression)	Object[]	Converts the expression into a JSON Object array.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>toBool(Expression)</code>	boolean	Converts the input value to a boolean.
<code>toNumber(Expression)</code>	number	Converts the input value to a number, if possible.
<code>toString(Expression)</code>	string	Converts the input value to a string.
<code>typeOf(Expression)</code>	string	Returns a string describing the type of the given value.

Color expressions

Color expressions make it easier to create and manipulate color values.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>color(int)</code>	color	Converts a color integer value into a color expression.
<code>rgb(Expression red, Expression green, Expression blue)</code> <code>rgb(number red, number green, number blue)</code>		Creates a color value from <i>red</i> , <i>green</i> , and <i>blue</i> components that must range between <code>0</code> and <code>255</code> , and an alpha component of <code>1</code> . If any component is out of range, the expression is an error.
<code>rgba(Expression red, Expression green, Expression blue, Expression alpha)</code> <code>rgba(number red, number green, number blue, number alpha)</code>		Creates a color value from <i>red</i> , <i>green</i> , <i>blue</i> components that must range between <code>0</code> and <code>255</code> , and an alpha component within a range of <code>0</code> and <code>1</code> . If any component is out of range, the expression is an error.
<code>toColor(Expression)</code>	color	Converts the input value to a color.
<code>toRgba(Expression)</code>	color	Returns a four-element array containing the input color's red, green, blue, and alpha components, in that order.

Example

The following example creates an RGB color value that has a *red* value of `255`, and *green* and *blue* values that are calculated by multiplying `2.5` by the value of the `temperature` property. As the temperature changes, the color will change to different shades of *red*.

```
BubbleLayer layer = new BubbleLayer(source,
    bubbleColor(
        //Create a RGB color value.
        rgb(
            //Set red value to 255. Wrap with literal expression since using expressions for other values.
            literal(255f),

            //Multiple the temperature by 2.5 and set the green value.
            product(literal(2.5f), get("temperature")),

            //Multiple the temperature by 2.5 and set the blue value.
            product(literal(2.5f), get("temperature"))
        )
    )
);
```

```
val layer = BubbleLayer(source,
    bubbleColor(
        //Create a RGB color value.
        rgb(
            //Set red value to 255. Wrap with literal expression since using expressions for other values.
            literal(255f),

            //Multiple the temperature by 2.5 and set the green value.
            product(literal(2.5f), get("temperature")),

            //Multiple the temperature by 2.5 and set the blue value.
            product(literal(2.5f), get("temperature"))
        )
    )
)
```

If all the color parameters are numbers, there is no need to wrap them with the `literal` expression. For example:

```
BubbleLayer layer = new BubbleLayer(source,
    bubbleColor(
        //Create a RGB color value.
        rgb(
            255f, //Set red value to 255.

            150f, //Set green value to 150.

            0f //Set blue value to 0.
        )
    )
);
```

```
val layer = BubbleLayer(source,
    bubbleColor(
        //Create a RGB color value.
        rgb(
            255f, //Set red value to 255.

            150f, //Set green value to 150.

            0f //Set blue value to 0.
        )
    )
)
```

TIP

String color values can be converted into a color using the `android.graphics.Color.parseColor` method. The following converts a hex color string into a color expression that can be used with a layer.

```
color(parseColor("#ff00ff"))
```

String operator expressions

String operator expressions perform conversion operations on strings such as concatenating and converting the case.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>concat(string...) concat(Expression...)</code>	string	Concatenates multiple strings together. Each value must be a string. Use the <code>toString</code> type expression to convert other value types to string if needed.
<code>downcase(string) downcase(Expression)</code>	string	Converts the specified string to lowercase.
<code>isSupportedScript(string) isSupportedScript(Expression)</code>	boolean	Determines if the input string uses a character set supported by the current font stack. For example: <code>isSupportedScript("𠁻𠁻𠁻𠁻𠁻𠁻𠁻𠁻𠁻")</code>
<code>resolvedLocale(Expression collator)</code>	string	Returns the IETF language tag of the locale being used by the provided collator. This can be used to determine the default system locale, or to determine if a requested locale was successfully loaded.
<code>upcase(string) upcase(Expression)</code>	string	Converts the specified string to uppercase.

Example

The following example converts the `temperature` property of the point feature into a string and then concatenates "^oF" to the end of it.

```
SymbolLayer layer = new SymbolLayer(source,
    textField(
        concat(Expression.toString(get("temperature")), literal("°F")),
    ),
    //Some additional style options.
    textOffset(new Float[] { 0f, -1.5f }),
    textSize(12f),
    textColor("white")
);
```

```

val layer = SymbolLayer(source,
    textField(
        concat(Expression.toString(get("temperature")), literal("°F"))
    ),
    //Some additional style options.
    textOffset(new Float[] { 0f, -1.5f }),
    textSize(12f),
    textColor("white")
)

```

The above expression renders a pin on the map with the text "64°F" overlaid on top of it as shown in the image below.



Interpolate and step expressions

Interpolate and step expressions can be used to calculate values along an interpolated curve or step function.

These expressions take in an expression that returns a numeric value as their input, for example

`get("temperature")`. The input value is evaluated against pairs of input and output values, to determine the value that best fits the interpolated curve or step function. The output values are called "stops". The input values for each stop must be a number and be in ascending order. The output values must be a number, and array of numbers, or a color.

Interpolate expression

An `interpolate` expression can be used to calculate a continuous, smooth set of values by interpolating between stop values. An `interpolate` expression that returns color values produces a color gradient in which result values are selected from. The `interpolate` expression has the following formats:

```

//Stops consist of two expressions.
interpolate(Expression.Interpolator interpolation, Expression number, Expression... stops)

//Stop expression wraps two values.
interpolate(Expression.Interpolator interpolation, Expression number, Expression.Stop... stops)

```

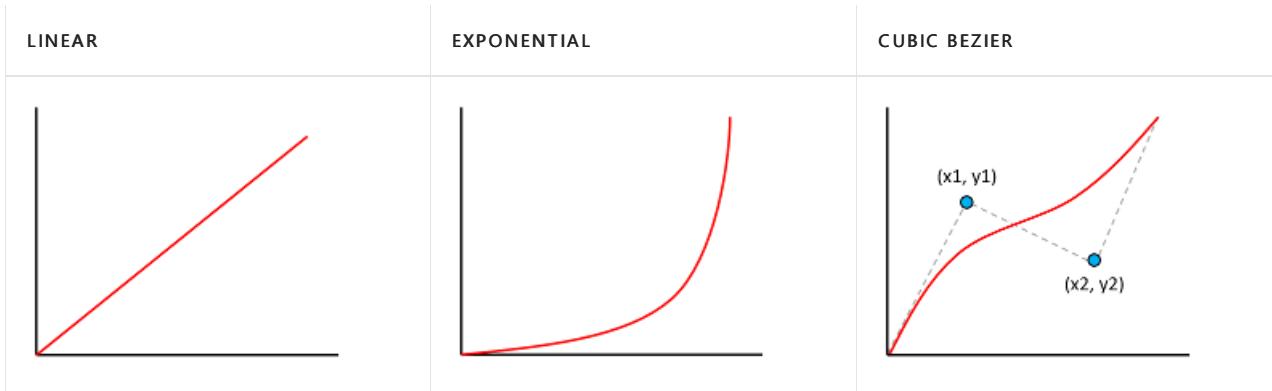
There are three types of interpolation methods that can be used in an `interpolate` expression:

NAME	DESCRIPTION
<code>linear()</code>	Interpolates linearly between the pair of stops.

NAME	DESCRIPTION
<code>exponential(number) exponential(Expression)</code>	Interpolates exponentially between the stops. A "base" is specified and controls the rate at which the output increases. Higher values make the output increase more towards the high end of the range. A "base" value close to 1 produces an output that increases more linearly.
<code>cubicBezier(number x1, number y1, number x2, number y2)</code> <code> </code> <code>cubicBezier(Expression x1, Expression y1, Expression x2, Expression y2)</code>	Interpolates using a cubic Bezier curve defined by the given control points.

The `stop` expression has the format `stop(stop, value)`.

Here is an example of what these different types of interpolations look like.



Example

The following example uses a `linear interpolate` expression to set the `bubbleColor` property of a bubble layer based on the `temperature` property of the point feature. If the `temperature` value is less than 60, "blue" will be returned. If it's between 60 and less than 70, yellow will be returned. If it's between 70 and less than 80, "orange" (`#FFA500`) will be returned. If it's 80 or greater, "red" will be returned.

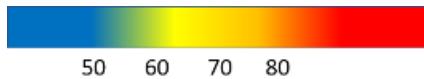
```
BubbleLayer layer = new BubbleLayer(source,
    bubbleColor(
        interpolate(
            linear(),
            get("temperature"),
            stop(50, color(Color.BLUE)),
            stop(60, color(Color.YELLOW)),
            stop(70, color(parseColor("#FFA500"))),
            stop(80, color(Color.RED))
        )
    )
);
```

```

val layer = BubbleLayer(source,
    bubbleColor(
        interpolate(
            linear(),
            get("temperature"),
            stop(50, color(Color.BLUE)),
            stop(60, color(Color.YELLOW)),
            stop(70, color(parseColor("#FFA500"))),
            stop(80, color(Color.RED))
        )
    )
)

```

The following image demonstrates how the colors are chosen for the above expression.



Step expression

A `step` expression can be used to calculate discrete, stepped result values by evaluating a [piecewise-constant function](#) defined by stops.

The `interpolate` expression has the following formats:

```

step(Expression input, Expression defaultOutput, Expression... stops)

step(Expression input, Expression defaultOutput, Expression.Stop... stops)

step(Expression input, number defaultOutput, Expression... stops)

step(Expression input, number defaultOutput, Expression.Stop... stops)

step(number input, Expression defaultOutput, Expression... stops)

step(number input, Expression defaultOutput, Expression.Stop... stops)

step(number input, number defaultOutput, Expression... stops)

step(number input, number defaultOutput, Expression.Stop... stops)

```

Step expressions return the output value of the stop just before the input value, or the first input value if the input is less than the first stop.

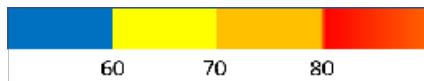
Example

The following example uses a `step` expression to set the `bubbleColor` property of a bubble layer based on the `temperature` property of the point feature. If the `temperature` value is less than 60, "blue" will be returned. If it's between 60 and less than 70, "yellow" will be returned. If it's between 70 and less than 80, "orange" will be returned. If it's 80 or greater, "red" will be returned.

```
BubbleLayer layer = new BubbleLayer(source,
    bubbleColor(
        step(
            get("temperature"),
            color(Color.BLUE),
            stop(60, color(Color.YELLOW)),
            stop(70, color(parseColor("#FFA500"))),
            stop(80, color(Color.RED))
        )
    )
);
```

```
val layer = BubbleLayer(source,
    bubbleColor(
        step(
            get("temperature"),
            color(Color.BLUE),
            stop(60, color(Color.YELLOW)),
            stop(70, color(parseColor("#FFA500"))),
            stop(80, color(Color.RED))
        )
    )
)
```

The following image demonstrates how the colors are chosen for the above expression.



Layer-specific expressions

Special expressions that only apply to specific layers.

Heat map density expression

A heat map density expression retrieves the heat map density value for each pixel in a heat map layer and is defined as `heatmapDensity`. This value is a number between `0` and `1`. It's used in combination with a `interpolation` or `step` expression to define the color gradient used to colorize the heat map. This expression can only be used in the `heatmapColor` option of the heat map layer.

TIP

The color at index 0, in an interpolation expression or the default color of a step color, defines the color of the area where there's no data. The color at index 0 can be used to define a background color. Many prefer to set this value to transparent or a semi-transparent black.

Example

This example uses a liner interpolation expression to create a smooth color gradient for rendering the heat map.

```
HeatMapLayer layer = new HeatMapLayer(source,
    heatmapColor(
        interpolate(
            linear(),
            heatmapDensity(),
            stop(0, color(Color.TRANSPARENT)),
            stop(0.01, color(Color.MAGENTA)),
            stop(0.5, color(parseColor("#fb00fb"))),
            stop(1, color(parseColor("#00c3ff")))
        )
    )
);
```

```
val layer = HeatMapLayer(source,
    heatmapColor(
        interpolate(
            linear(),
            heatmapDensity(),
            stop(0, color(Color.TRANSPARENT)),
            stop(0.01, color(Color.MAGENTA)),
            stop(0.5, color(parseColor("#fb00fb"))),
            stop(1, color(parseColor("#00c3ff")))
        )
    )
)
```

In addition to using a smooth gradient to colorize a heat map, colors can be specified within a set of ranges by using a `step` expression. Using a `step` expression for colorizing the heat map visually breaks up the density into ranges that resembles a contour or radar style map.

```
HeatMapLayer layer = new HeatMapLayer(source,
    heatmapColor(
        step(
            heatmapDensity(),
            color(Color.TRANSPARENT),
            stop(0.01, color(parseColor("#000080"))),
            stop(0.25, color(parseColor("#000080"))),
            stop(0.5, color(Color.GREEN)),
            stop(0.5, color(Color.YELLOW)),
            stop(1, color(Color.RED))
        )
    )
);
```

```
val layer = HeatMapLayer(source,
    heatmapColor(
        step(
            heatmapDensity(),
            color(Color.TRANSPARENT),
            stop(0.01, color(parseColor("#000080"))),
            stop(0.25, color(parseColor("#000080"))),
            stop(0.5, color(Color.GREEN)),
            stop(0.5, color(Color.YELLOW)),
            stop(1, color(Color.RED))
        )
    )
)
```

For more information, see the [Add a heat map layer](#) documentation.

Line progress expression

A line progress expression retrieves the progress along a gradient line in a line layer and is defined as `lineProgress()`. This value is a number between 0 and 1. It's used in combination with an `interpolation` or `step` expression. This expression can only be used with the `strokeGradient` option of the line layer.

NOTE

The `strokeGradient` option of the line layer requires the `lineMetrics` option of the data source to be set to `true`.

Example

This example uses the `lineProgress()` expression to apply a color gradient to the stroke of a line.

```
LineLayer layer = new LineLayer(source,
    strokeGradient(
        interpolate(
            linear(),
            lineProgress(),
            stop(0, color(Color.BLUE)),
            stop(0.1, color(Color.argb(255, 65, 105, 225))), //Royal Blue
            stop(0.3, color(Color.CYAN)),
            stop(0.5, color(Color.argb(255,0, 255, 0))), //Lime
            stop(0.7, color(Color.YELLOW)),
            stop(1, color(Color.RED)))
    )
);
```

```
val layer = LineLayer(source,
    strokeGradient(
        interpolate(
            linear(),
            lineProgress(),
            stop(0, color(Color.BLUE)),
            stop(0.1, color(Color.argb(255, 65, 105, 225))), //Royal Blue
            stop(0.3, color(Color.CYAN)),
            stop(0.5, color(Color.argb(255,0, 255, 0))), //Lime
            stop(0.7, color(Color.YELLOW)),
            stop(1, color(Color.RED)))
    )
);
```

[See live example](#)

Text field format expression

The `format` expression can be used with the `textField` option of the symbol layer to provide mixed text formatting. This expression takes in one or more `formatEntry` expressions that specify a string and set of `formatOptions` to append to the text field.

EXPRESSION	DESCRIPTION
<code>format(Expression...)</code>	Returns formatted text containing annotations for use in mixed-format text-field entries.

EXPRESSION	DESCRIPTION
<pre>formatEntry(Expression text) formatEntry(Expression text, Expression.FormatOption... expressionOptions) formatEntry(String text) formatEntry(String text, Expression.FormatOption... formatOptions)</pre>	Returns a formatted string entry for use in the <code>format</code>

The following format options available are:

EXPRESSION	DESCRIPTION
<pre>formatFontSize(number) formatFontSize(Expression)</pre>	Specifies the scaling factor for the font size. If specified, this value will override the <code>textSize</code> property for the individual string.
<pre>formatTextFont(string[]) formatTextFont(Expression)</pre>	Specifies a color to apply to a text when rendering.

Example

The following example formats the text field by adding a bold font and scaling up the font size of the `title` property of the feature. This example also adds the `subTitle` property of the feature on a newline, with a scaled down font size.

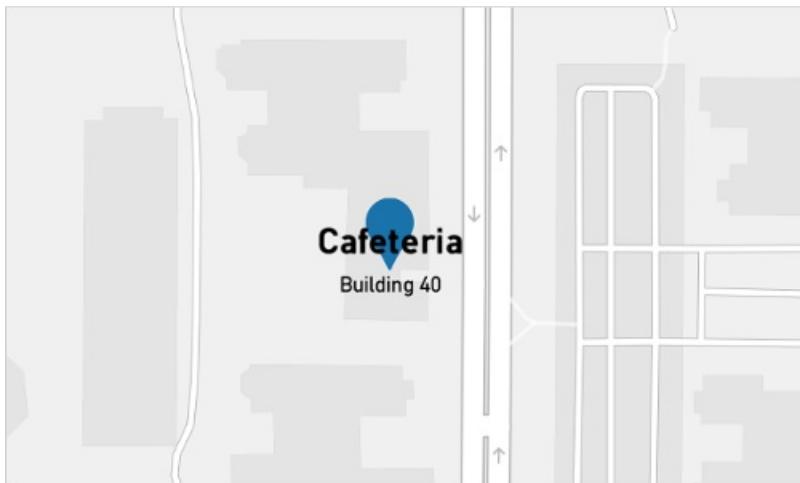
```
SymbolLayer layer = new SymbolLayer(source,
    textField(
        format(
            //Bold the title property and scale its font size up.
            formatEntry(
                get("title"),
                formatTextFont(new String[] { "StandardFont-Bold" }),
                formatFontSize(1.25)),
            //Add a new line without any formatting.
            formatEntry("\n"),
            //Scale the font size down of the subTitle property.
            formatEntry(
                get("subTitle"),
                formatFontSize(0.75))
        )
    );
}
```

```

val layer = SymbolLayer(source,
    textField(
        format(
            //Bold the title property and scale its font size up.
            formatEntry(
                get("title"),
                formatTextFont(arrayOf("StandardFont-Bold")),
                formatFontSize(1.25)),
            //Add a new line without any formatting.
            formatEntry("\n"),
            //Scale the font size down of the subTitle property.
            formatEntry(
                get("subTitle"),
                formatFontSize(0.75))
        )
    )
)

```

This layer will render the point feature as shown in the image below:



Zoom expression

A `zoom` expression is used to retrieve the current zoom level of the map at render time and is defined as `zoom()`. This expression returns a number between the minimum and maximum zoom level range of the map. The Azure Maps interactive map controls for web and Android support 25 zoom levels, numbered 0 through 24. Using the `zoom` expression allows styles to be modified dynamically as the zoom level of the map is changed. The `zoom` expression may only be used with `interpolate` and `step` expressions.

Example

By default, the radii of data points rendered in the heat map layer have a fixed pixel radius for all zoom levels. As the map is zoomed, the data aggregates together and the heat map layer looks different. A `zoom` expression can be used to scale the radius for each zoom level such that each data point covers the same physical area of the map. It will make the heat map layer look more static and consistent. Each zoom level of the map has twice as many pixels vertically and horizontally as the previous zoom level. Scaling the radius, such that it doubles with each zoom level, will create a heat map that looks consistent on all zoom levels. It can be accomplished using the `zoom` expression with a `base 2 exponential interpolation` expression, with the pixel radius set for the minimum zoom level and a scaled radius for the maximum zoom level calculated as `2 * Math.pow(2, minZoom - maxZoom)` as shown below.

```

HeatMapLayer layer = new HeatMapLayer(source,
    heatmapRadius(
        interpolate(
            exponential(2),
            zoom(),
            //For zoom level 1 set the radius to 2 pixels.
            stop(1, 2),
            //Between zoom level 1 and 19, exponentially scale the radius from 2 pixels to 2 * (maxZoom - minZoom)^2 pixels.
            stop(19, 2 * Math.pow(2, 19 - 1))
        )
    );
);

```

```

val layer = HeatMapLayer(source,
    heatmapRadius(
        interpolate(
            exponential(2),
            zoom(),
            //For zoom level 1 set the radius to 2 pixels.
            stop(1, 2),
            //Between zoom level 1 and 19, exponentially scale the radius from 2 pixels to 2 * (maxZoom - minZoom)^2 pixels.
            stop(19, 2 * Math.pow(2, 19 - 1))
        )
    );
)
;

```

Variable binding expressions

Variable binding expressions store the results of a calculation in a variable. So, that the calculation results can be referenced elsewhere in an expression multiple times. It is a useful optimization for expressions that involve many calculations.

EXPRESSION	RETURN TYPE	DESCRIPTION
<code>let(Expression... input)</code>		Stores one or more values as variables for use by the <code>var</code> expression in the child expression that returns the result.
<code>var(Expression expression) var(string variableName)</code>	Object	References a variable that was created using the <code>let</code> expression.

Example

This example uses an expression that calculates the revenue relative to temperature ratio and then uses a `case` expression to evaluate different boolean operations on this value. The `let` expression is used to store the revenue relative to temperature ratio, so that it only needs to be calculated once. The `var` expression references this variable as often as needed without having to recalculate it.

```

BubbleLayer layer = new BubbleLayer(source,
    bubbleColor(
        let(
            //Divide the point features `revenue` property by the `temperature` property and store it in a
            variable called `ratio`.
            literal("ratio"), division(get("revenue"), get("temperature")),

            //Evaluate the child expression in which the stored variable will be used.
            switchCase(
                //Check to see if the ratio is less than 100, return 'red'.
                lt(var("ratio"), 100), color(Color.RED),

                //Check to see if the ratio is less than 200, return 'green'.
                lt(var("ratio"), 200), color(Color.GREEN),

                //Return `blue` for values greater or equal to 200.
                color(Color.BLUE)
            )
        )
    )
);

```

```

val layer = BubbleLayer(source,
    bubbleColor(
        let(
            //Divide the point features `revenue` property by the `temperature` property and store it in a
            variable called `ratio`.
            literal("ratio"), division(get("revenue"), get("temperature")),

            //Evaluate the child expression in which the stored variable will be used.
            switchCase(
                //Check to see if the ratio is less than 100, return 'red'.
                lt(var("ratio"), 100), color(Color.RED),

                //Check to see if the ratio is less than 200, return 'green'.
                lt(var("ratio"), 200), color(Color.GREEN),

                //Return `blue` for values greater or equal to 200.
                color(Color.BLUE)
            )
        )
    )
)

```

Next steps

Learn more about the layers that support expressions:

[Add a symbol layer](#)

[Add a bubble layer](#)

[Add a line layer](#)

[Add a polygon layer](#)

[Add a heat map](#)

Interact with the map (Android SDK)

3/26/2021 • 4 minutes to read • [Edit Online](#)

This article shows you how to use the maps events manager.

Interact with the map

The map manages all events through its `events` property. The following table lists all of the supported map events.

EVENT	EVENT HANDLER FORMAT	DESCRIPTION
<code>OnCameraIdle</code>	<code>()</code>	<p>Fired after the last frame rendered before the map enters an "idle" state:</p> <ul style="list-style-type: none">• No camera transitions are in progress.• All currently requested tiles have loaded.• All fade/transition animations have completed.
<code>OnCameraMove</code>	<code>()</code>	Fired repeatedly during an animated transition from one view to another, as the result of either user interaction or methods.
<code>OnCameraMoveCanceled</code>	<code>()</code>	Fired when a movement request to the camera has been canceled.
<code>OnCameraMoveStarted</code>	<code>(int reason)</code>	<p>Fired just before the map begins a transition from one view to another, as the result of either user interaction or methods. The <code>reason</code> argument of the event listener returns an integer value that provides details of how the camera movement was initiated. The following list outlines the possible reasons:</p> <ul style="list-style-type: none">• 1: Gesture• 2: Developer animation• 3: API Animation
<code>OnClick</code>	<code>(double lat, double lon): boolean</code>	Fired when the map is pressed and released at the same point on the map. This event handler returns a boolean value indicating if the event should be consumed or passed further to other event listeners.

EVENT	EVENT HANDLER FORMAT	DESCRIPTION
OnFeatureClick	(List<Feature>): boolean	Fired when the map is pressed and released at the same point on a feature. This event handler returns a boolean value indicating if the event should be consumed or passed further to other event listeners.
OnLayerAdded	(Layer layer)	Fired when a layer is added to the map.
OnLayerRemoved	(Layer layer)	Fired when a layer is removed from the map.
OnLoaded	()	Fired immediately after all necessary resources have been downloaded and the first visually complete rendering of the map has occurred.
OnLongClick	(double lat, double lon): boolean	Fired when the map is pressed, held for a moment, and then released at the same point on the map. This event handler returns a boolean value indicating if the event should be consumed or passed further to other event listeners.
OnLongFeatureClick	(List<Feature>): boolean	Fired when the map is pressed, held for a moment, and then released at the same point on a feature. This event handler returns a boolean value indicating if the event should be consumed or passed further to other event listeners.
OnReady	(AzureMap map)	Fired when the map initially is loaded or when the app orientation change and the minimum required map resources are loaded and the map is ready to be programmatically interacted with.
OnSourceAdded	(Source source)	Fired when a <code>DataSource</code> or <code>VectorTileSource</code> is added to the map.
OnSourceRemoved	(Source source)	Fired when a <code>DataSource</code> or <code>VectorTileSource</code> is removed from the map.
OnStyleChange	()	Fired when the map's style loads or changes.

The following code shows how to add the `OnClick`, `OnFeatureClick`, and `OnCameraMove` events to the map.

```

map.events.add((OnClick) (lat, lon) -> {
    //Map clicked.

    //Return true indicating if event should be consumed and not passed further to other listeners
    //registered afterwards, false otherwise.
    return true;
});

map.events.add((OnFeatureClick) (features) -> {
    //Feature clicked.

    //Return true indicating if event should be consumed and not passed further to other listeners
    //registered afterwards, false otherwise.
    return true;
});

map.events.add((OnCameraMove) () -> {
    //Map camera moved.
});

```

```

map.events.add(OnClick { lat: Double, lon: Double ->
    //Map clicked.

    //Return true indicating if event should be consumed and not passed further to other listeners
    //registered afterwards, false otherwise.
    return false
})

map.events.add(OnFeatureClick { features: List<Feature?>? ->
    //Feature clicked.

    //Return true indicating if event should be consumed and not passed further to other listeners
    //registered afterwards, false otherwise.
    return false
})

map.events.add(OnCameraMove {
    //Map camera moved.
})

```

For more information, see the [Navigating the map](#) documentation on how to interact with the map and trigger events.

Scope feature events to layer

When adding the `OnFeatureClick` or `OnLongFeatureClick` events to the map, a layer instance or layer ID can be passed in as a second parameter. When a layer is passed in, the event will only fire if the event occurs on that layer. Events scoped to layers are supported by the symbol, bubble, line, and polygon layers.

```

//Create a data source.
DataSource source = new DataSource();
map.sources.add(source);

//Add data to the data source.
source.add(Point.fromLngLat(0, 0));

//Create a layer and add it to the map.
BubbleLayer layer = new BubbleLayer(source);
map.layers.add(layer);

//Add a feature click event to the map and pass the layer ID to limit the event to the specified layer.
map.events.add((OnFeatureClick) (features) -> {
    //One or more features clicked.

    //Return true indicating if event should be consumed and not passed further to other listeners
    //registered afterwards, false otherwise.
    return true;
}, layer);

//Add a long feature click event to the map and pass the layer ID to limit the event to the specified layer.
map.events.add((OnLongFeatureClick) (features) -> {
    //One or more features long clicked.

    //Return true indicating if event should be consumed and not passed further to other listeners
    //registered afterwards, false otherwise.
    return true;
}, layer);

```

```

//Create a data source.
val source = DataSource()
map.sources.add(source)

//Add data to the data source.
source.add(Point.fromLngLat(0, 0))

//Create a layer and add it to the map.
val layer = BubbleLayer(source)
map.layers.add(layer)

//Add a feature click event to the map and pass the layer ID to limit the event to the specified layer.
map.events.add(
    OnFeatureClick { features: List<Feature?>? ->
        //One or more features clicked.

        //Return true indicating if event should be consumed and not passed further to other listeners
        //registered afterwards, false otherwise.
        return false
    },
    layer
)

//Add a long feature click event to the map and pass the layer ID to limit the event to the specified layer.
map.events.add(
    OnLongFeatureClick { features: List<Feature?>? ->
        //One or more features long clicked.

        //Return true indicating if event should be consumed and not passed further to other listeners
        //registered afterwards, false otherwise.
        return false
    },
    layer
)

```

Next steps

See the following articles for full code examples:

[Navigating the map](#)

[Add a symbol layer](#)

[Add a bubble layer](#)

[Add a line layer](#)

[Add a polygon layer](#)

Getting started with the Azure Maps Power BI visual

3/5/2021 • 6 minutes to read • [Edit Online](#)

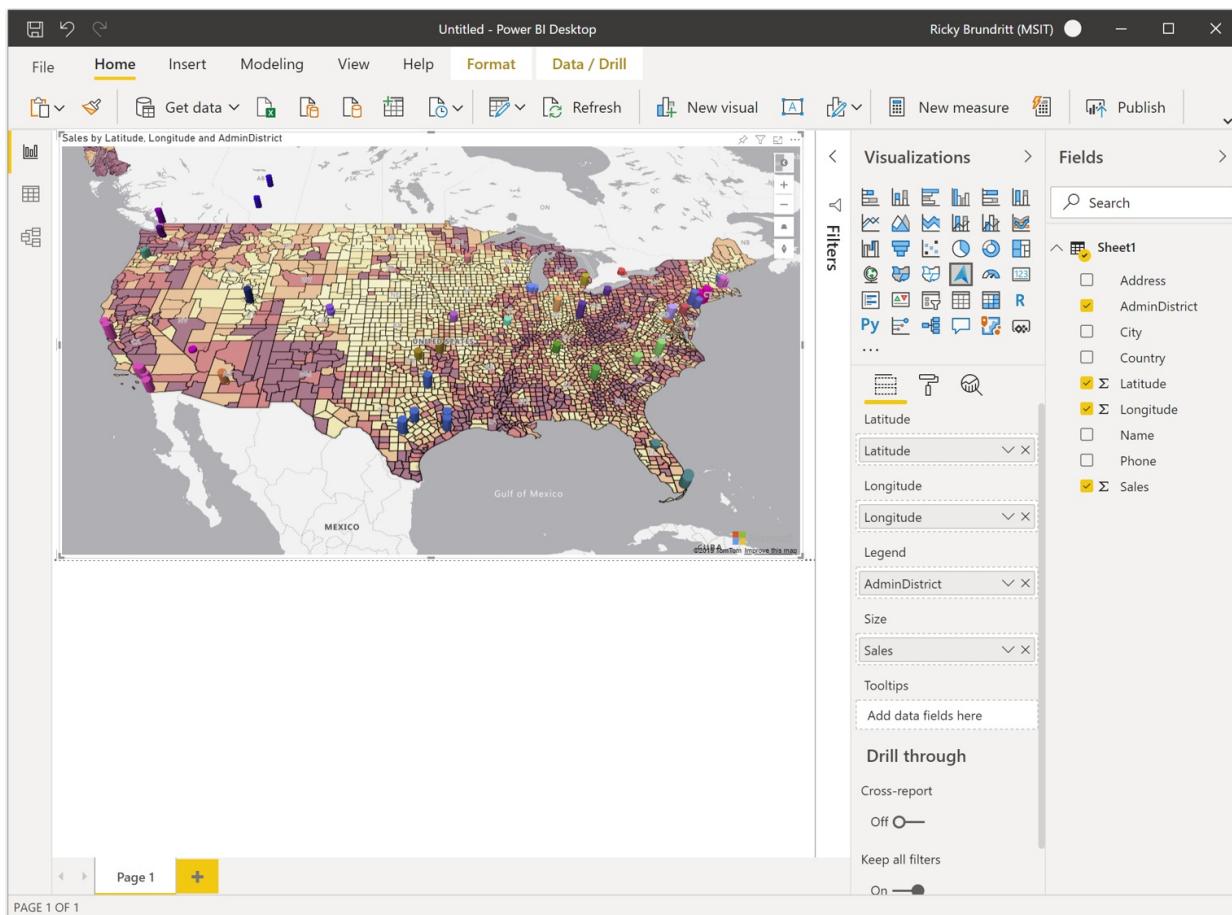
APPLIES TO: ✓ Power BI service for *consumers* ✓ Power BI service for designers & developers ✓
Power BI Desktop ✗ Requires Pro or Premium license

This article shows how to use the Microsoft Azure Maps visual for Power BI.

NOTE

This visual can be created and viewed in both Power BI Desktop and the Power BI service. The steps and illustrations in this article are from Power BI Desktop.

The Azure Maps visual for Power BI provides a rich set of data visualizations for spatial data on top of a map. It is estimated that over 80% of business data has a location context. The Azure Maps visual can be used to gain insights into how this location context relates to and influences your business data.



What is sent to Azure?

The Azure Maps visual connects to cloud service hosted in Azure to retrieve location data such as map images and coordinates that are used to create the map visualization.

- Details about the area the map is focused on are sent to Azure to retrieve images needed to render the map canvas (also known as map tiles).
- Data in the Location, Latitude, and Longitude buckets may be sent to Azure to retrieve map coordinates (a process called geocoding).

- Telemetry data may be collected on the health of the visual (i.e. crash reports), if the telemetry option in Power BI is enabled.

Other than the scenarios described above, no other data overlaid on the map is sent to the Azure Maps servers. All rendering of data happens locally within the client.

You, or your administrator, may need to update your firewall to allow access to the Azure Maps platform which uses the following URL.

```
https://atlas.microsoft.com
```

To learn more, about privacy and terms of use related to the Azure Maps visual see [Microsoft Azure Legal Information](#).

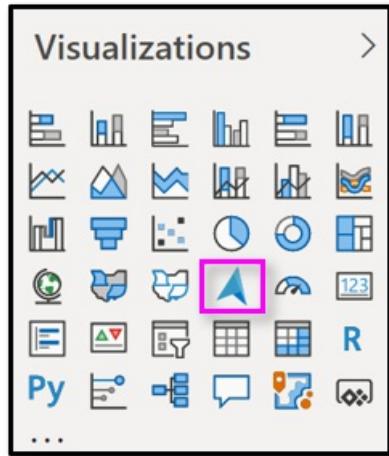
Azure Maps visual (Preview) behavior and requirements

There are a few considerations and requirements for **Azure Maps** visual. :

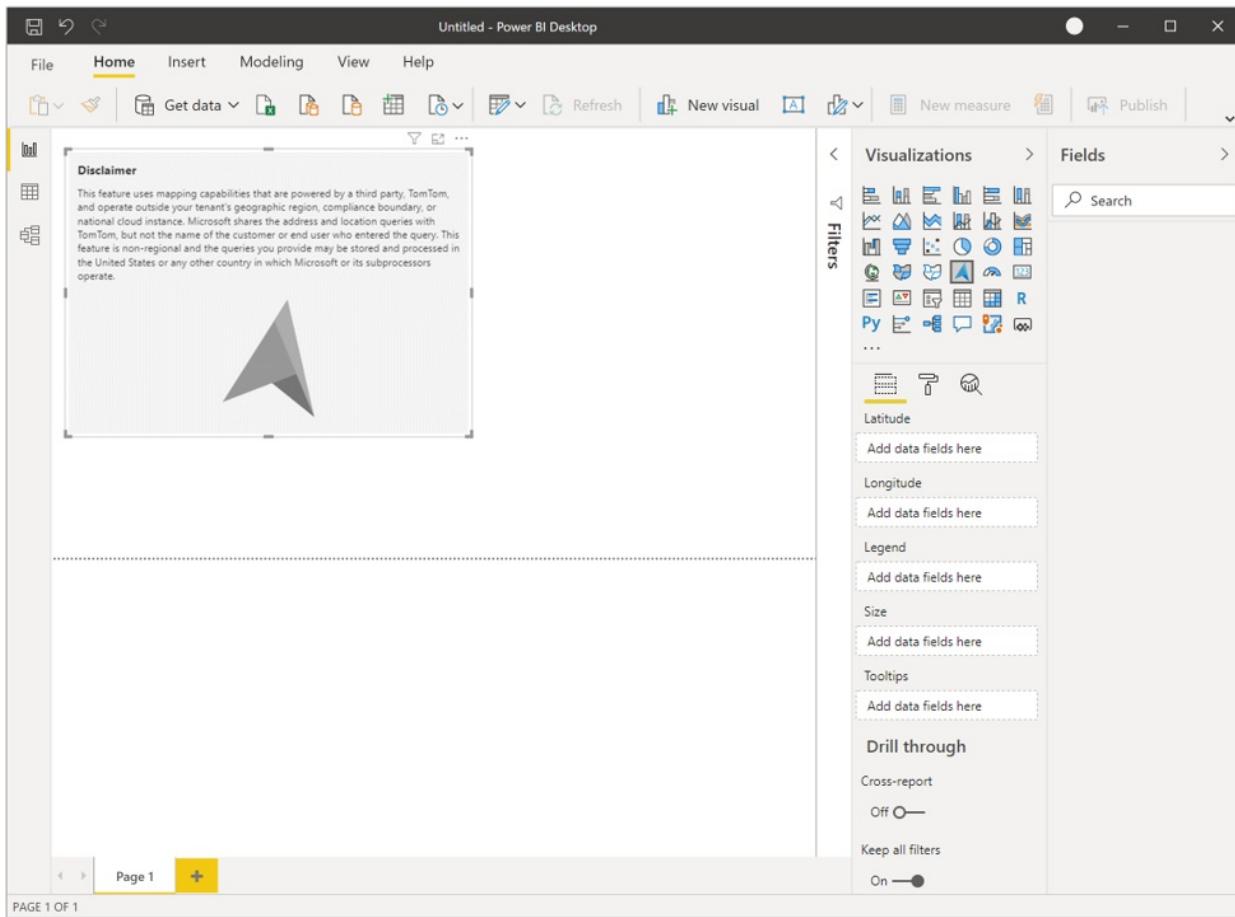
- The **Azure Maps** visual (Preview) must be enabled in Power BI Desktop. To enable **Azure Maps** visual, select **File > Options and Settings > Options > Preview features**, then select the **Azure Maps Visual** checkbox. If the Azure Maps visual is not available after doing this, it's likely that a tenant admin switch in the Admin Portal needs to be enabled.
- The data set must have fields that contain **latitude** and **longitude** information. Geocoding of location fields will be added in a future update.
- The built-in legend control for Power BI does not currently appear in this preview. It will be added in a future update.

Use the Azure Maps visual (Preview)

Once the **Azure Maps** visual is enabled, select the **Azure Maps** icon from the **Visualizations** pane.

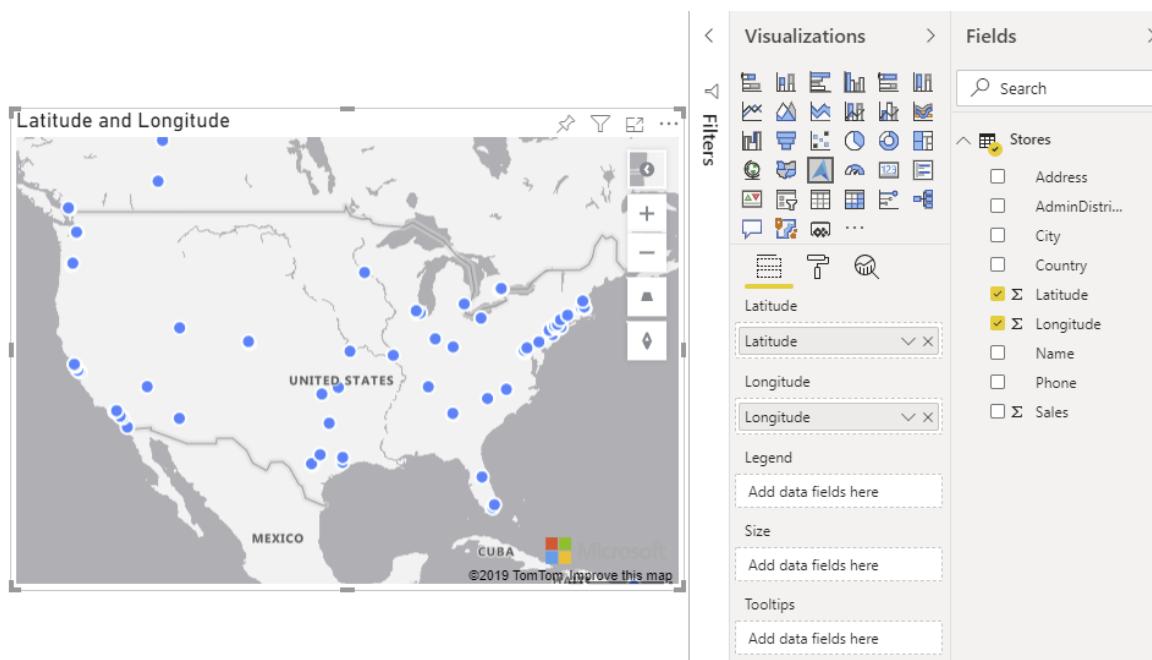


Power BI creates an empty Azure Maps visual design canvas. While in preview, an additional disclaimer is displayed.

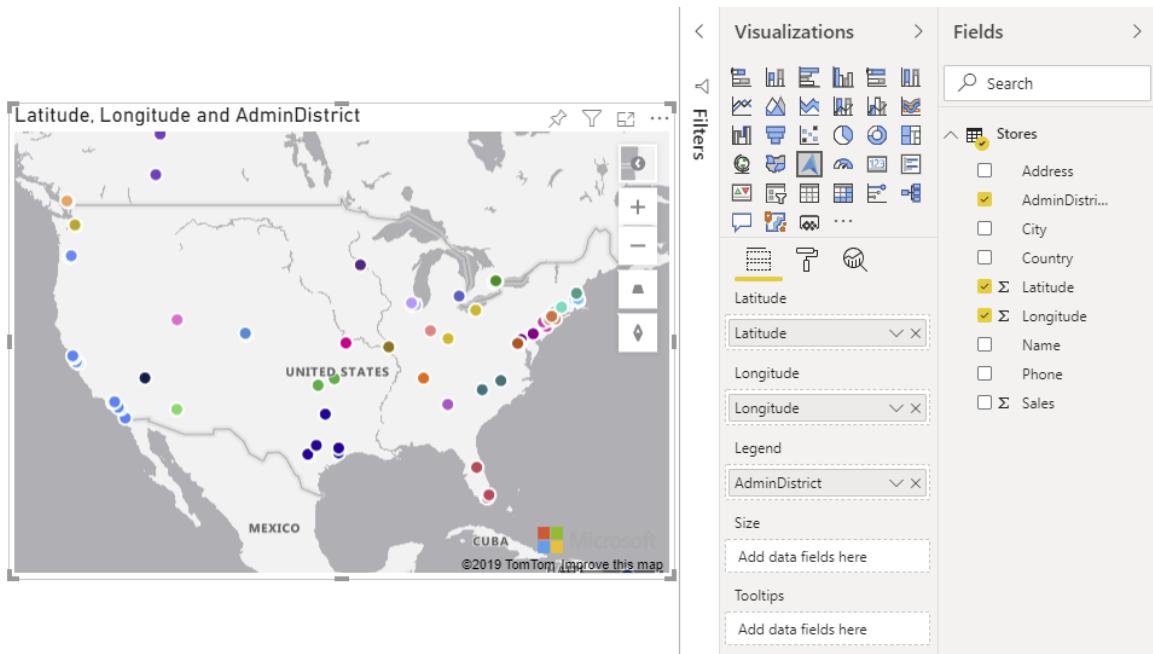


Take the following steps to load the Azure Maps visual:

1. In the **Fields** pane, drag data fields that contain latitude and longitude coordinate information into the **Latitude** and/or **Longitude** buckets. This is the minimal data needed to load the Azure Maps visual.



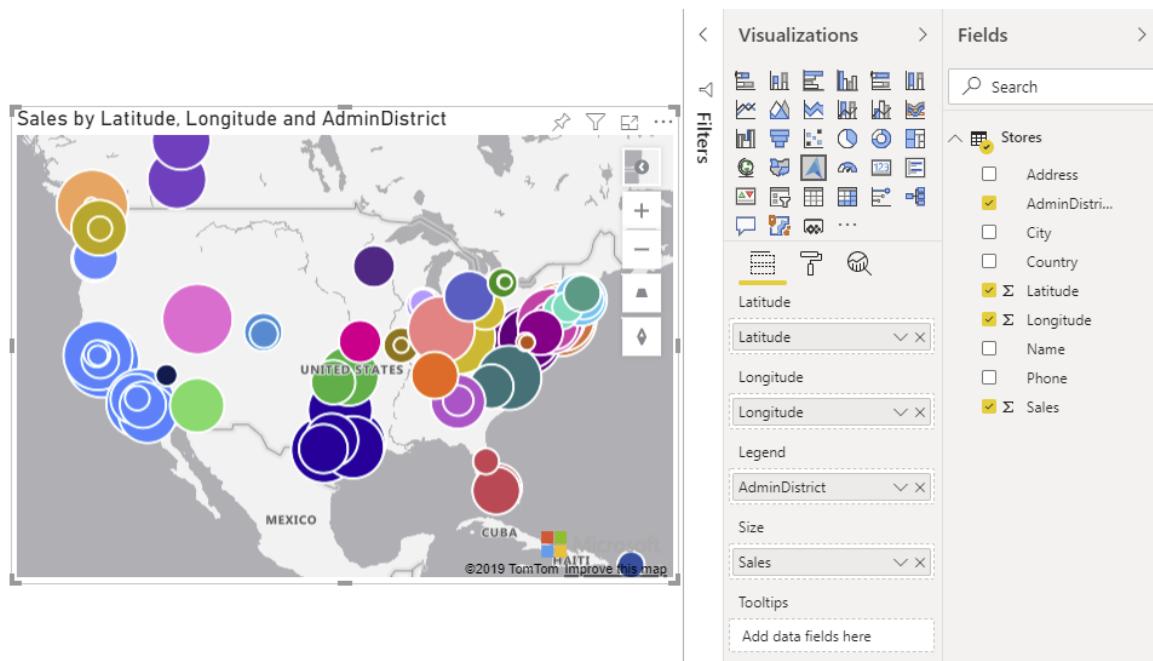
2. To color the data based on categorization, drag a categorical field into the **Legend** bucket of the **Fields** pane. In this example, we're using the **AdminDistrict** column (also known as state or province).



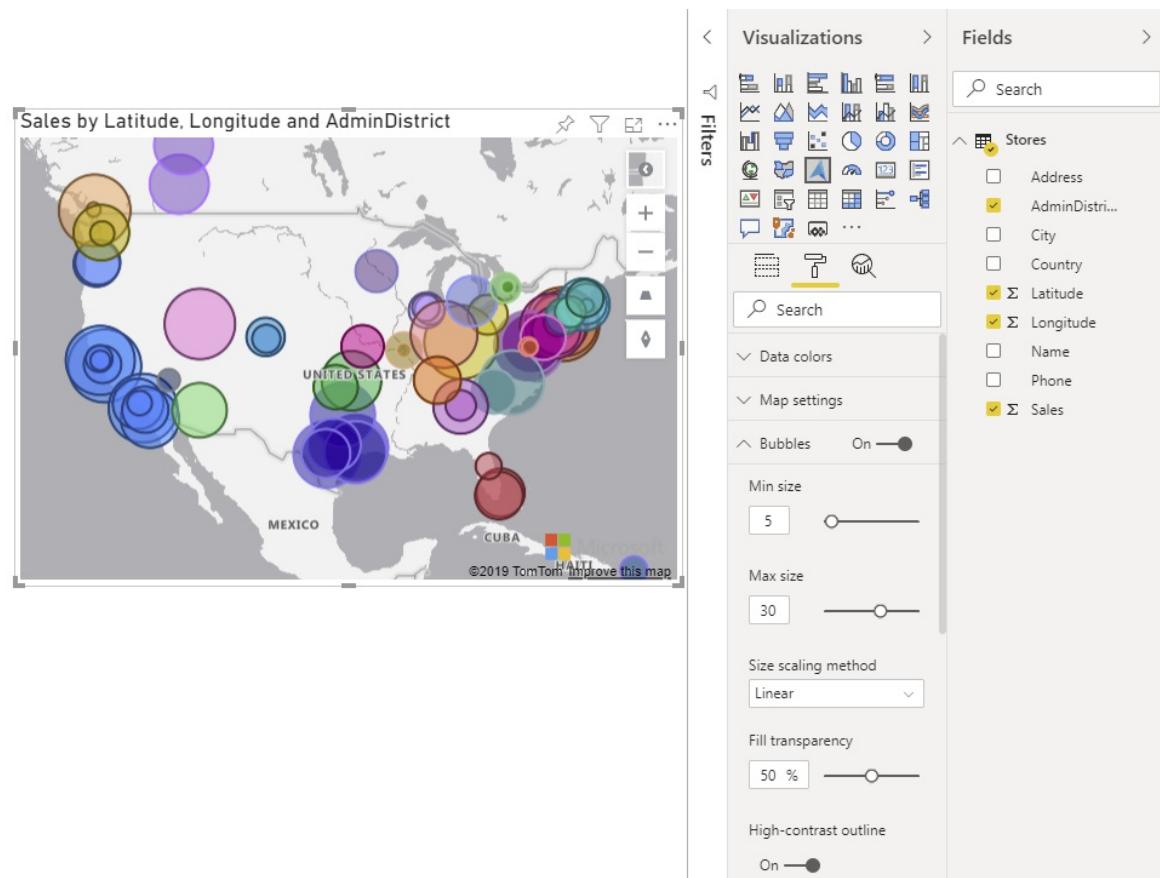
NOTE

The built-in legend control for Power BI does not currently appear in this preview. It will be added in a future update.

- To scale the data relatively, drag a measure into the **Size** bucket of the **Fields** pane. In this example, we're using **Sales** column.



- Use the options in the **Format** pane to customize how data is rendered. The following image is the same map as above, but with the bubble layers fill transparency option set to 50% and the high-contrast outline option enabled.



Fields pane buckets

The following data buckets are available in the **Fields** pane of the Azure Maps visual.

FIELD	DESCRIPTION
Latitude	The field used to specify the latitude value of the data points. Latitude values should be between -90 and 90 in decimal degrees format.
Longitude	The field used to specify the longitude value of the data points. Longitude values should be between -180 and 180 in decimal degrees format.
Legend	The field used to categorize the data and assign a unique color for data points in each category. When this bucket is filled, a Data colors section will appear in the Format pane which allows adjustments to the colors.
Size	The measure used for relative sizing of data points on the map.
Toolips	Additional data fields that are displayed in tooltips when shapes are hovered.

Map settings

The **Map settings** section of the Format pane provide options for customizing how the map is displayed and reacts to updates.

SETTING	DESCRIPTION
Auto zoom	Automatically zooms the map into the data loaded through the Fields pane of the visual. As the data changes, the map will update its position accordingly. When the slider is in the Off position, additional map view settings are displayed for the default map view.
World wrap	Allows the user to pan the map horizontally infinitely.
Style picker	Adds a button to the map that allows the report readers to change the style of the map.
Navigation controls	Adds buttons to the map as another method to allow the report readers to zoom, rotate, and change the pitch of the map. For more information, see this document on Navigating the map for details on all the different ways users can navigate the map.
Map style	The style of the map. For more information, see this document for more information on supported map styles .

Map view settings

If the **Auto zoom** slider is in the **Off** position, the following settings are displayed and allow the user to specify the default map view information.

SETTING	DESCRIPTION
Zoom	The default zoom level of the map. Can be a number between 0 and 22.
Center latitude	The default latitude at the center of the map.
Center longitude	The default longitude at the center of the map.
Heading	The default orientation of the map in degrees, where 0 is north, 90 is east, 180 is south, and 270 is west. Can be any number between 0 and 360.
Pitch	The default tilt of the map in degrees between 0 and 60, where 0 is looking straight down at the map.

Considerations and Limitations

The Azure Maps visual is available in the following services and applications:

SERVICE/APP	AVAILABILITY
Power BI Desktop	Yes
Power BI service (app.powerbi.com)	Yes
Power BI mobile applications	Yes

SERVICE/APP	AVAILABILITY
Power BI publish to web	No
Power BI Embedded	No
Power BI service embedding (PowerBI.com)	Yes

Support for additional Power BI services/apps will be added in future updates.

Where is Azure Maps available?

At this time, Azure Maps is currently available in all countries and regions except the following:

- China
- South Korea

For coverage details for the different Azure Maps services that power this visual, see the [Geographic coverage information](#) document.

Which web browsers are supported by the Azure Maps visual?

See this documentation for information on [Azure Maps Web SDK supported browsers](#).

How many data points can I visualize?

This visual supports up to 30,000 data points.

Can addresses or other location strings be used in this visual?

The initial preview of this visual only supports latitude and longitude values in decimal degrees. A future update will add support for addresses and other location strings.

Next steps

Learn more about the Azure Maps Power BI visual:

[Understanding layers in the Azure Maps Power BI visual](#)

[Manage the Azure Maps visual within your organization](#)

Customize the visual:

[Tips and tricks for color formatting in Power BI](#)

[Customize visualization titles, backgrounds, and legends](#)

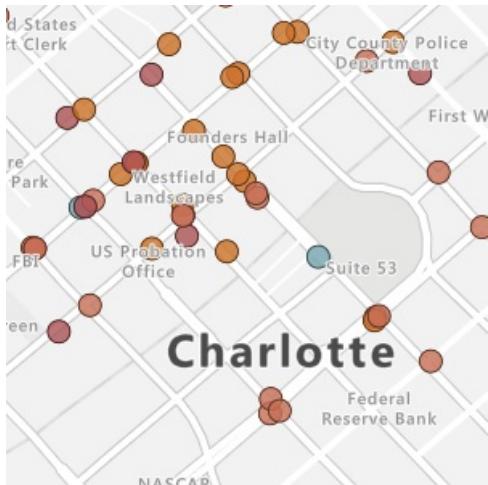
Understanding layers in the Azure Maps Power BI visual

11/2/2020 • 2 minutes to read • [Edit Online](#)

There are two types of layers available in the Azure Maps visual. The first type focuses on rendering data that is passed into the **Fields** pane of the visual and consist of the following layers, let's call these data rendering layers.

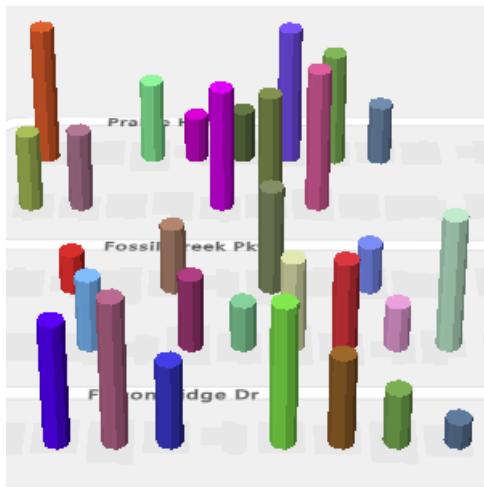
Bubble layer

Renders points as scaled circles on the map.



Bar chart layer

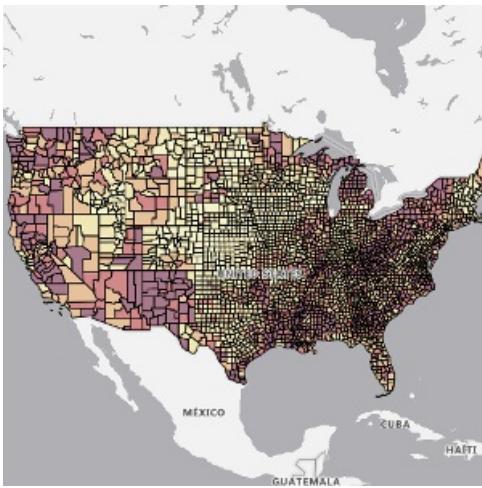
Renders points as 3D bars on the map.



The second type of layer connects addition external sources of data to map to provide more context and consists of the following layers.

Reference layer

Overlay an uploaded GeoJSON file on top of the map.



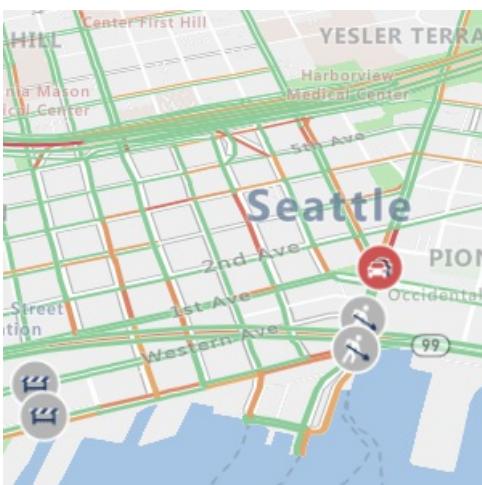
Tile layer

Overlay a custom tile layer on top of the map.



Traffic layer

Overlay real-time traffic information on the map.



All the data rendering layers, as well as the **Tile layer**, have options for min and max zoom levels that are used to specify a zoom level range these layers should be displayed at. This allows one type of rendering layer to be used at one zoom level and a transition to another rendering layer at another zoom level.

These layers also have an option to be positioned relative to other layers in the map. When multiple data rendering layers are used, the order in which they are added to the map determines their relative layering order when they have the same **Layer position** value.

General layer settings

The general layer section of the **Format** pane are common settings that apply to the layers that are connected to the Power BI dataset in the **Fields** pane (Bubble layer, Bar chart).

SETTING	DESCRIPTION
Unselected transparency	The transparency of shapes that are not selected, when one or more shapes are selected.
Show zeros	Specifies if points that have a size value of zero should be shown on the map using the minimum radius.
Show negatives	Specifies if absolute value of negative size values should be plotted.
Min data value	The minimum value of the input data to scale against. Good for clipping outliers.
Max data value	The maximum value of the input data to scale against. Good for clipping outliers.

Next steps

Change how your data is displayed on the map:

[Add a bubble layer](#)

[Add a bar chart layer](#)

Add more context to the map:

[Add a reference layer](#)

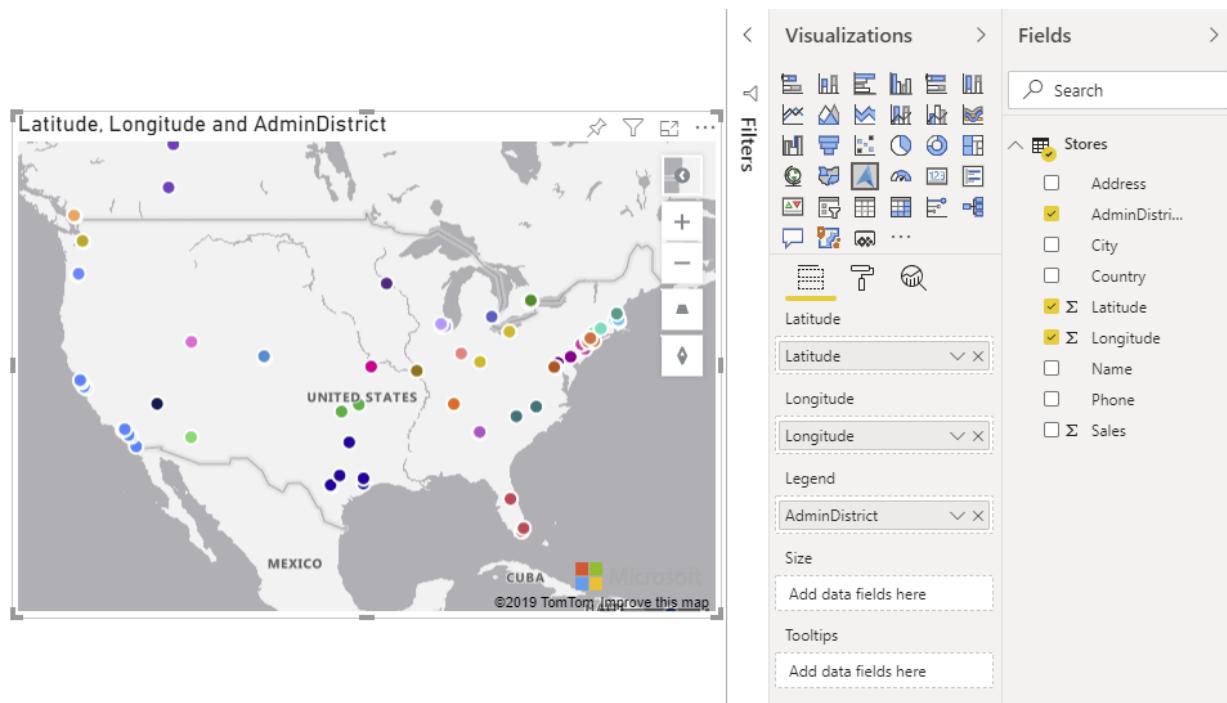
[Add a tile layer](#)

[Show real-time traffic](#)

Add a bubble layer

11/2/2020 • 3 minutes to read • [Edit Online](#)

The **Bubble layer** renders location data as scaled circles on the map.



Initially all bubbles have the same fill color. If a field is passed into the **Legend** bucket of the **Fields** pane, the bubbles will be colored based on their categorization. The outline of the bubbles is white by default but can be changed to a new color or by enabling the high-contrast outline option. The **High-contrast outline** option dynamically assigns an outline color that is a high-contrast variant of the fill color. This helps to ensure the bubbles are clearly visible regardless of the style of the map. The following are the primary settings in the **Format** pane that are available in the **Bubble layer** section.

SETTING	DESCRIPTION
Size	The size of each bubble. This option is hidden when a field is passed into the Size bucket of the Fields pane. Additional options will appear as outlined in the Bubble size scaling topic further down in this article.
Fill color	Color of each bubble. This option is hidden when a field is passed into the Legend bucket of the Fields pane and a separate Data colors section will appear in the Format pane.
Fill transparency	Transparency of each bubble.
High-contrast outline	Makes the outline color contrast with the fill color for better accessibility by using a high-contrast variant of the fill color.
Outline color	Color that outlines the bubble. This option is hidden when the High-contrast outline option is enabled.

SETTING	DESCRIPTION
Outline transparency	Transparency of the outline.
Outline width	Width of the outline in pixels.
Blur	Amount of blur applied to the outline. A value of 1 blurs the bubbles such that only the center point has no transparency. A value of 0 applies any blur effect.
Pitch alignment	Specifies how the bubbles look when the map is pitched. <ul style="list-style-type: none"> • Viewport - Bubbles appear on their edge on the map relative to viewport. (default) • Map - Bubbles are rendered flat on the surface of the map.
Zoom scale	Amount the bubbles should scale relative to the zoom level. A zoom scale of one means no scaling. Large values will make bubbles smaller when zoomed out and larger when zoomed in. This helps to reduce the clutter on the map when zoomed out, yet ensures points stand out more when zoomed in. A value of 1 does not apply any scaling.
Min zoom	Minimum zoom level tiles are available.
Max zoom	Maximum zoom level tiles are available.
Layer position	Specifies the position of the layer relative to other map layers.

Bubble size scaling

If a field is passed into the **Size** bucket of the **Fields** pane, the bubbles will be scaled relatively to the measure value of each data point. The **Size** option in the **Bubble layer** section of the **Format** pane will disappear when a field is passed into the **Size** bucket, as the bubbles will have their radii scaled between a min and max value. The following options will appear in the **Bubble layer** section of the **Format** pane when a **Size** bucket has a field specified.

SETTING	DESCRIPTION
Min size	Minimum bubble size when scaling the data.
Max size	Maximum bubble size when scaling the data.
Size scaling method	Scaling algorithm used to determine relative bubble size. <ul style="list-style-type: none"> • Linear - Range of input data linearly mapped to the min and max size. (default) • Log - Range of input data logarithmically mapped to the min and max size. • Cubic-Bezier - Specify X1, Y1, X2, Y2 values of a Cubic-Bezier curve to create a custom scaling method.

When the **Size scaling method** is set to **Log**, the following options will be made available.

SETTING	DESCRIPTION
Log scale	The logarithmic scale to apply when calculating the size of the bubbles.

When the **Size scaling method** is set to **Cubic-Bezier**, the following options will be made available to customize the scaling curve.

SETTING	DESCRIPTION
X1	X1 parameter of a cubic Bezier curve.
Y1	Y2 parameter of a cubic Bezier curve.
X2	Y1 parameter of a cubic Bezier curve.
Y2	Y2 parameter of a cubic Bezier curve.

TIP

<https://cubic-bezier.com/> has a handy tool for creating the parameters for Cubic-Bezier curves.

Next steps

Change how your data is displayed on the map:

[Add a bar chart layer](#)

Add more context to the map:

[Add a reference layer](#)

[Add a tile layer](#)

[Show real-time traffic](#)

Customize the visual:

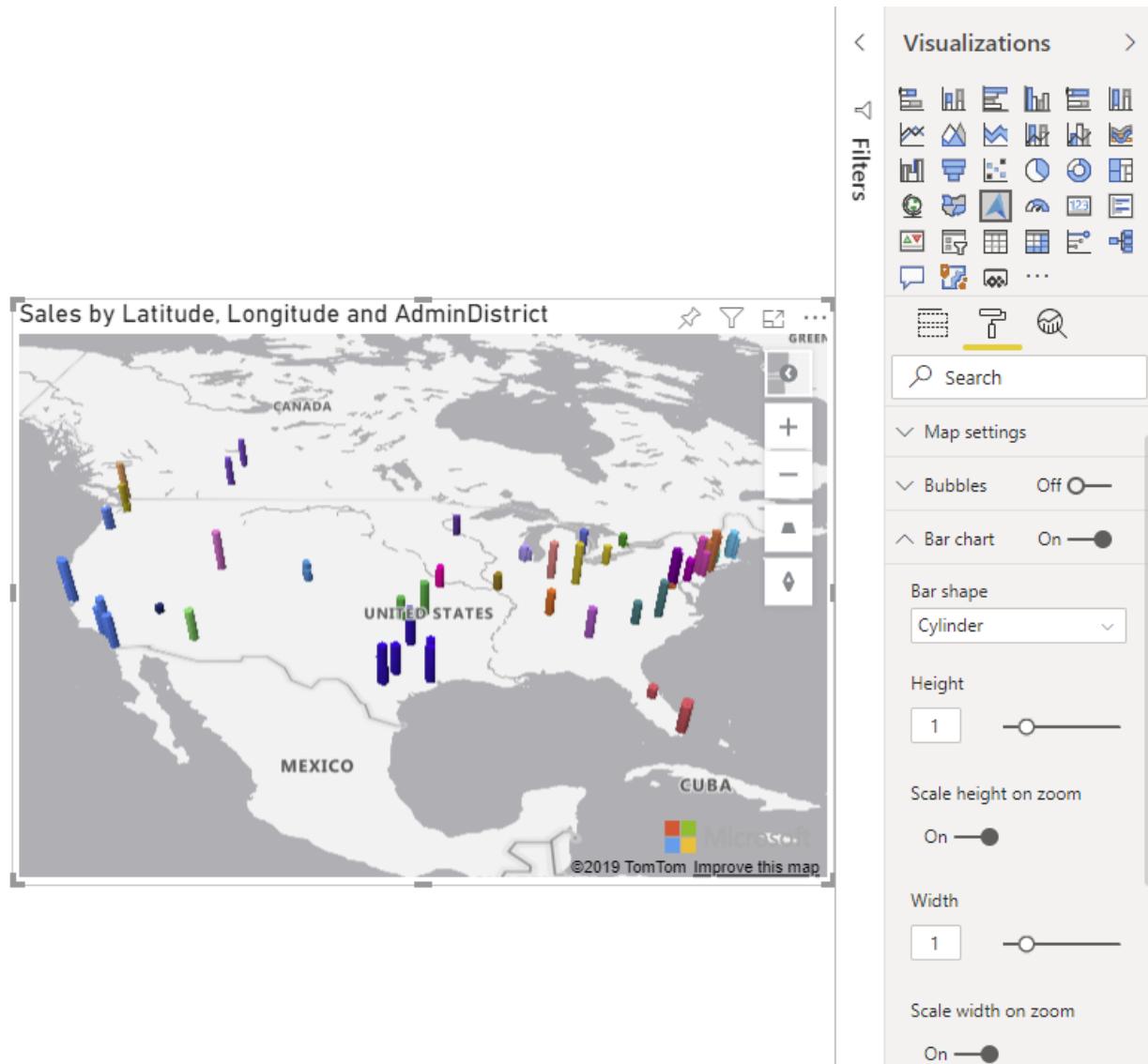
[Tips and tricks for color formatting in Power BI](#)

[Customize visualization titles, backgrounds, and legends](#)

Add a bar chart layer

11/2/2020 • 2 minutes to read • [Edit Online](#)

The **Bar chart layer** is useful for taking data to the next dimension by allowing visualization of location data as 3D bars or cylinders on the map. Similar to the bubble layer, the bar chart later can easily visualize two metrics at the same time using color and relative height. In order for the bars to have height, a measure needs to be added to the **Size** bucket of the **Fields** pane. If a measure is not provided, bars with no height show as flat squares or circles depending on the **Bar shape** option.



Users can tilt and rotate the map to view your data from different perspectives. The map can be tilted or pitched using one of the following methods.

- Turn on the **Navigation controls** option in the **Map settings** of the **Format** pane. This will add a button to tilt the map.
- Press the right mouse button down and drag the mouse up or down.
- Using a touch screen, touch the map with two fingers and drag them up or down together.
- With the map focused, hold the **Shift** key, and press the **Up** or **Down** arrow keys.

The map can be rotated using one of the following methods.

- Turn on the **Navigation controls** option in the **Map settings** of the **Format** pane. This will add a button to

rotate the map.

- Press the right mouse button down and drag the mouse left or right.
- Using a touch screen, touch the map with two fingers and rotate.
- With the map focused, hold the **Shift** key, and press the **Left** or **Right arrow** keys.

The following are all settings in the **Format** pane that are available in the **Bar chart layer** section.

SETTING	DESCRIPTION
Bar shape	The shape of the 3D bar. <ul style="list-style-type: none">• Box – Bars rendered as rectangular boxes.• Cylinder – Bars rendered as cylinders.
Height	The height of each bar. If a field is passed into the Size bucket of the Fields pane, bars will be scaled relative to this height value.
Scale height on zoom	Specifies if the height of the bars should scale relative to the zoom level.
Width	The width of each bar.
Scale width on zoom	Specifies if the width of the bars should scale relative to the zoom level.
Fill color	Color of each bar. This option is hidden when a field is passed into the Legend bucket of the Fields pane and a separate Data colors section will appear in the Format pane.
Transparency	Transparency of each bar.
Min zoom	Minimum zoom level tiles are available.
Max zoom	Maximum zoom level tiles are available.
Layer position	Specifies the position of the layer relative to other map layers.

NOTE

If the bars have a small width value and the **Scale width on zoom** option is disabled, they may disappear when zoomed out a lot as their rendered width would be less than a pixel in size. However, when the **Scale width on zoom** option is enabled, additional calculations are performed when the zoom level changes which can impact performance of large data sets.

Next steps

Add more context to the map:

[Add a reference layer](#)

[Add a tile layer](#)

[Show real-time traffic](#)

Customize the visual:

[Tips and tricks for color formatting in Power BI](#)

[Customize visualization titles, backgrounds, and legends](#)

Add a reference layer

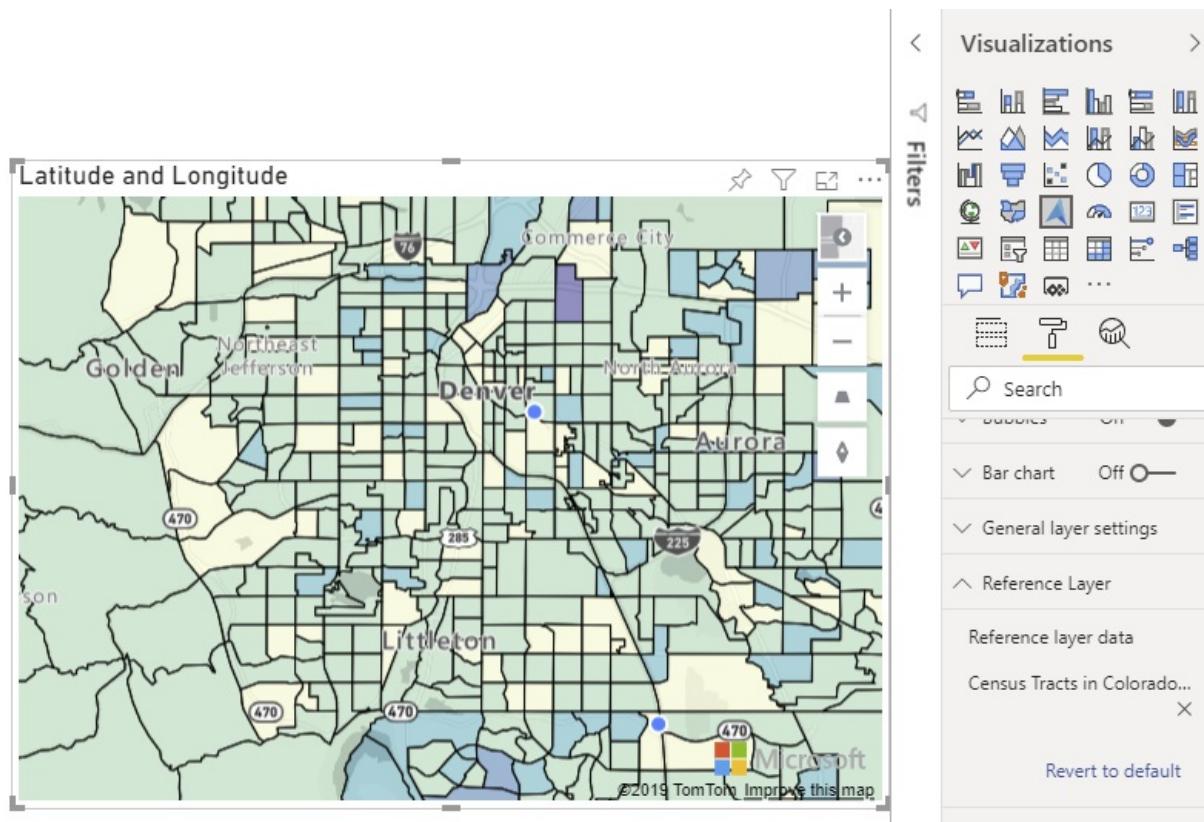
11/2/2020 • 2 minutes to read • [Edit Online](#)

The reference layer feature lets a secondary spatial dataset be uploaded to the visual and overlaid on the map to provide addition context. This dataset is hosted by Power BI and must be a [GeoJSON file](#) with a `.json` or `.geojson` file extension.

To add a [GeoJSON](#) file as a reference layer, go to the **Format** pane, expand the **Reference layer** section, and press the **+ Add local file** button.

After a [GeoJSON](#) file is added to the reference layer, the name of the file will appear in place of the **+ Add local file** button with an X beside it. Press the X button to remove the data from the visual and delete the [GeoJSON](#) file from Power BI.

The following map displays [2016 census tracts for Colorado](#), colored by population.



The following are all settings in the **Format** pane that are available in the **Reference layer** section.

SETTING	DESCRIPTION
Reference layer data	The data GeoJSON file to upload to the visual as an additional layer within the map. The + Add local file button opens a file dialog the user can use to select a GeoJSON file that has a <code>.json</code> or <code>.geojson</code> file extension.

NOTE

In this preview of the Azure Maps visual, the reference layer will only load the first 5,000 shape features to the map. This limit will be increased in a future update.

Styling data in a reference layer

Properties can be added to each feature within the GeoJSON file to customize how it is styled on the map. This feature uses the simple data layer feature in the Azure Maps Web SDK. For more information, see this document on [supported style properties](#). Custom icon images are not supported within the Azure Maps visual as a security precaution.

The following is an example of a GeoJSON point feature that sets its displayed color to red.

```
{  
    "type": "Feature",  
    "geometry": {  
        "type": "Point",  
        "coordinates": [-122.13284, 47.63699]  
    },  
    "properties": {  
        "color": "red"  
    }  
}
```

Next steps

Add more context to the map:

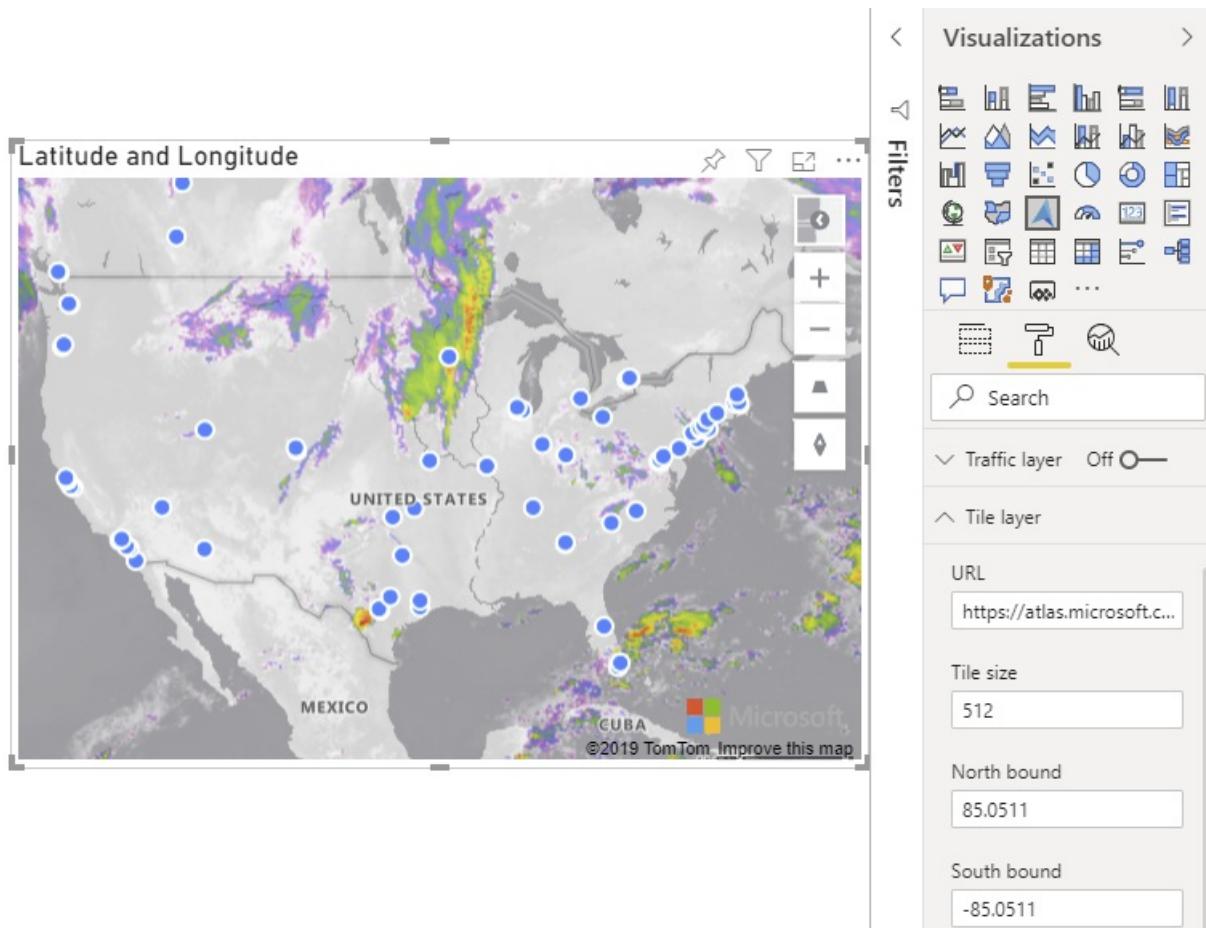
[Add a tile layer](#)

[Show real-time traffic](#)

Add a tile layer

11/2/2020 • 2 minutes to read • [Edit Online](#)

The tile layer feature, like the reference layer feature, allows additional data to be overlaid on the map to provide more context. Tile layers allow you to superimpose images on top of the Azure Maps base map tiles. This is a great way to overlay large or complex datasets such as imagery from drones, or millions of rows of data.



A tile layer loads in tiles from a server. These images can either be pre-rendered or dynamically rendered. Pre-rendered images are stored like any other image on a server using a naming convention that the tile layer understands. Dynamically rendered images use a service to load the images close to real time. Tile layers are a great way to visualize large datasets on the map. Not only can a tile layer be generated from an image, vector data can also be rendered as a tile layer too.

The bounding box and zoom range of where a tile service is available can be passed as settings to limit where tiles are requested. This is both a performance enhancement for both the visual and the tile service. Below is an overview of all settings available in the **Format** pane that are available in the **Tile layer** section.

SETTING	DESCRIPTION
Url	A formatted URL pointing to a tile service.
Tile size	An integer value that specifies both the width and height dimensions of the tiles.

SETTING	DESCRIPTION
North bound	Northern latitude of the bounding box where tiles are available.
South bound	Southern latitude of the bounding box where tiles are available.
East bound	Eastern longitude of the bounding box where tiles are available.
West bound	Western longitude of the bounding box where tiles are available.
Transparency	Transparency of the tile layer.
Is TMS	Tile Map Services, a specification that reverses the Y coordinate axis of the tile layer.
Min zoom	Minimum zoom level tiles are available.
Max zoom	Maximum zoom level tiles are available.
Layer position	Specifies the position of the layer relative to other map layers.

Tile URL formatting

There are three different tile service naming conventions supported by the Azure Maps visual:

- **X, Y, Zoom notation** - X is the column, Y is the row position of the tile in the tile grid, and the Zoom notation a value based on the zoom level.
- **Quadkey notation** - Combines x, y, and zoom information into a single string value. This string value becomes a unique identifier for a single tile.
- **Bounding Box** - Specify an image in the Bounding box coordinates format: `{west},{south},{east},{north}`. This format is commonly used by [Web Mapping Services \(WMS\)](#).

The tile URL an https URL to a tile URL template that uses the following parameters:

- `{x}` - X position of the tile. Also needs `{y}` and `{z}`.
- `{y}` - Y position of the tile. Also needs `{x}` and `{z}`.
- `{z}` - Zoom level of the tile. Also needs `{x}` and `{y}`.
- `{quadkey}` - Tile `quadkey` identifier based on the Bing Maps tile system naming convention.
- `{bbox-epsg-3857}` - A bounding box string with the format `{west},{south},{east},{north}` in the EPSG 3857 spatial reference system.

As an example, the following is a formatted tile URL for the [weather radar tile service](#) in Azure Maps. Note that `[subscription-key]` is a placeholder for your Azure Maps subscription key.

```
https://atlas.microsoft.com/map/tile?zoom={z}&x={x}&y={y}&tilesetId=microsoft.weather.radar.main&api-version=2.0&subscription-key=[subscription-key]
```

For more information on Azure Maps tiling system, see [Zoom levels and tile grid](#).

Next steps

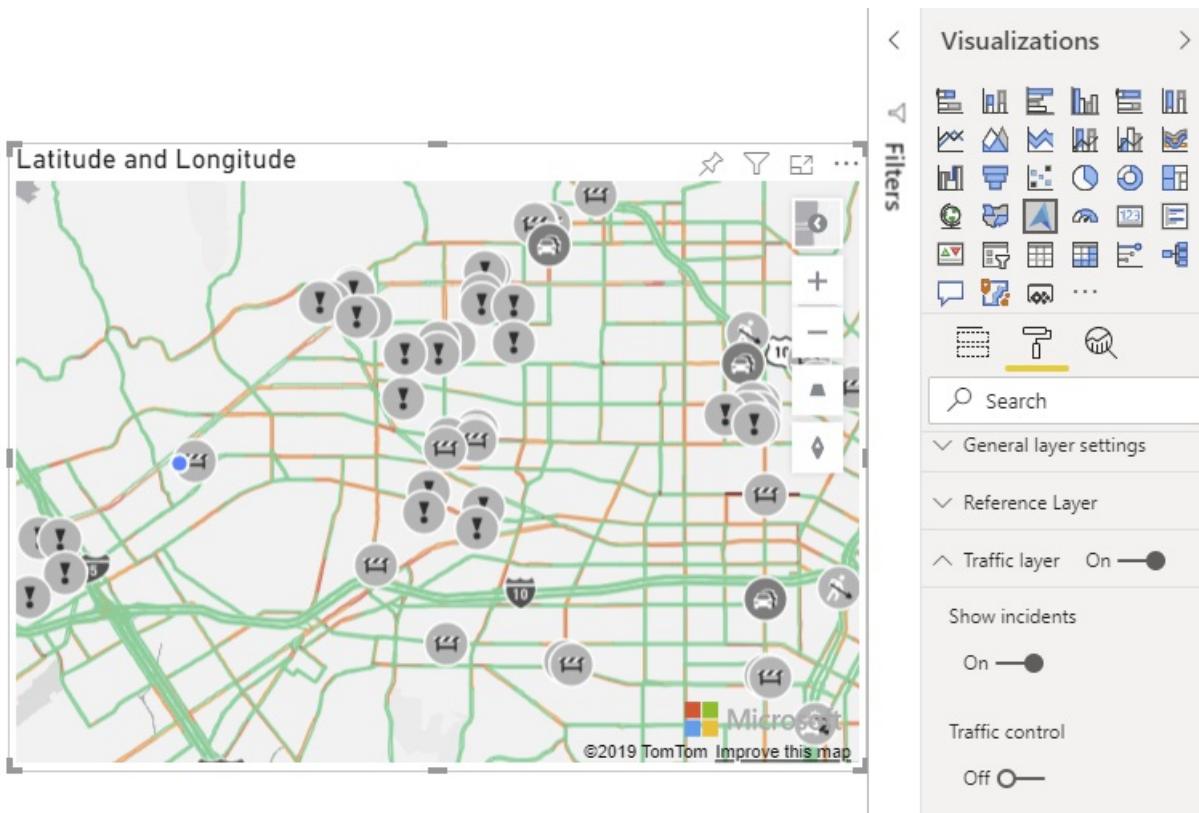
Add more context to the map:

[Show real-time traffic](#)

Show real-time traffic

11/2/2020 • 2 minutes to read • [Edit Online](#)

The traffic layer feature overlays real-time traffic data on top of the map. To enable this feature, move the **Traffic layer** slider in the **Format** pane to the **On** position. This will overlay traffic flow data as color coded roads.



The following settings are available in the **Traffic layer** section.

SETTING	DESCRIPTION
Show incidents	Specifies if traffic incidents, such as road closures and construction, should be displayed on the map.
Traffic control	Adds a button to the map that allows report readers to turn the traffic layer on or off.

Next steps

Learn more about the Azure Maps Power BI visual:

[Understanding layers in the Azure Maps Power BI visual](#)

[Manage the Azure Maps visual within your organization](#)

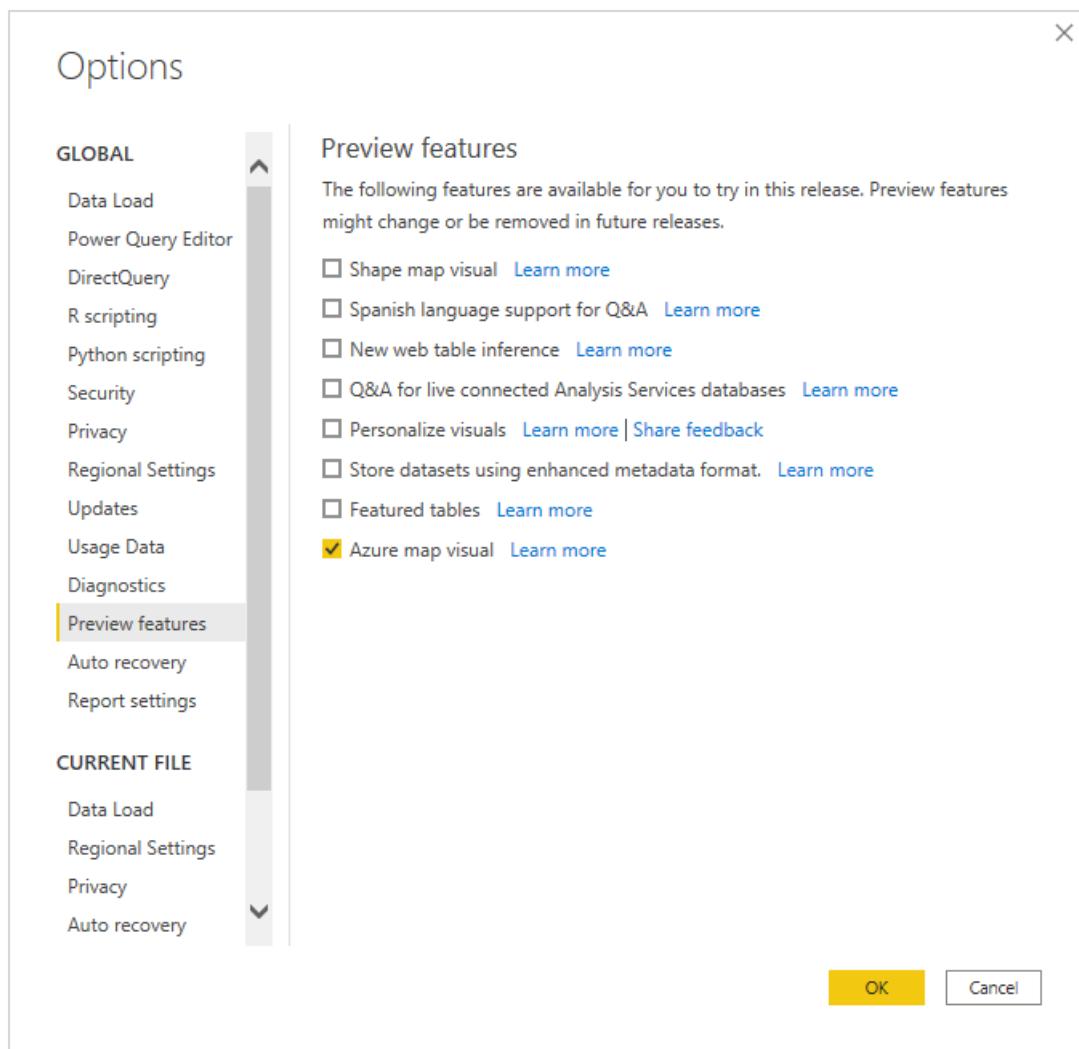
Manage the Azure Maps visual within your organization

11/2/2020 • 2 minutes to read • [Edit Online](#)

Power BI provides the ability for designers and tenant administrators to manage the use of the Azure Maps visual. Below you will find steps each role can take.

Designer options

In Power BI Desktop, designers can disable the Azure Maps visual on the security tab. Select **File > Options and settings** and then select **Options > Preview features**. When disabled, Azure Maps will not load by default.



Tenant admin options

In PowerBI.com, tenant administrators can turn off the Azure Maps visual for all users. Select **Settings > Admin Portal > Tenant settings**. When disabled, Power BI will no longer display the Azure Maps visual in the visualizations pane.

Admin portal

Usage metrics

Users

Audit logs

Tenant settings

Capacity settings

Embed Codes

Organizational visuals

Dataflow settings

Workspaces

Custom branding

Use Azure Maps visual

Disabled for the entire organization

Users in the organization can use the Azure Maps visualization.

 Enabled

 By selecting "Enabled", you agree that Azure Maps visuals may use Azure services located outside of your Power BI tenant's geographic region, compliance boundary, or national cloud instance. This feature uses mapping capabilities that are powered by a third party, TomTom, and operate outside your tenant's geographic region, compliance boundary, or national cloud instance. Microsoft shares the address and location queries with TomTom, but not the name of the customer or end user who entered the query. This feature is non-regional and the queries you provide may be stored and processed in the United States or any other country in which Microsoft or its subprocessors operate. Use of Azure Maps is subject to the following [terms](#). [Learn more](#)

[Apply](#)

[Cancel](#)

 This setting applies to the entire organization

Next steps

Learn more about the Azure Maps Power BI visual:

[Understanding layers in the Azure Maps Power BI visual](#)

Provide data feedback to Azure Maps

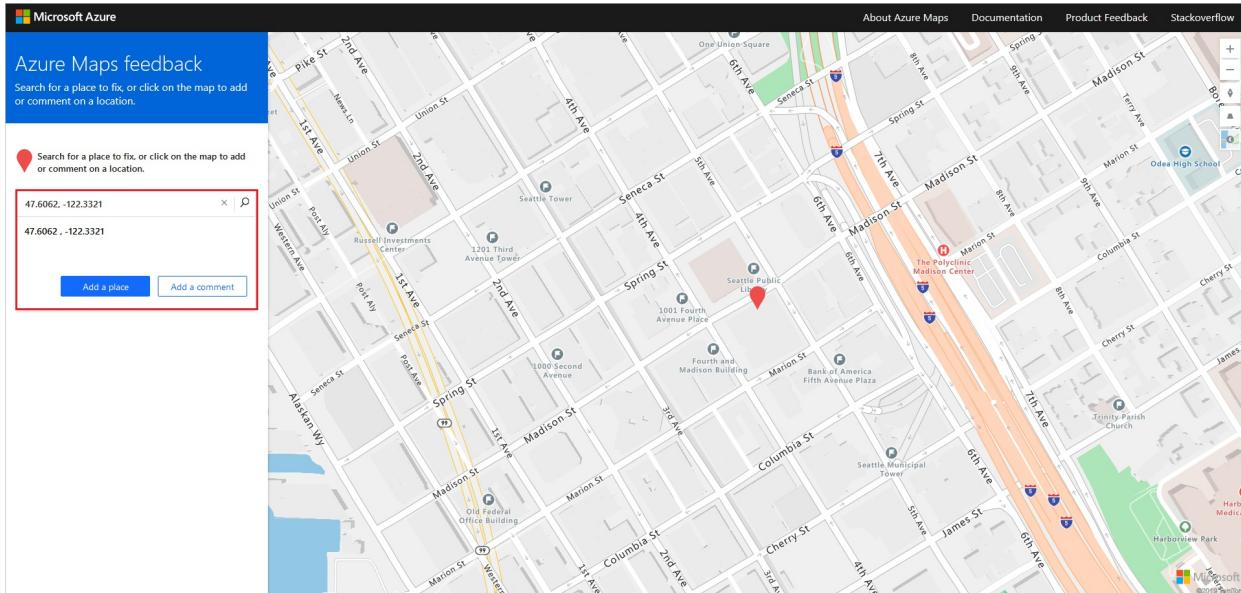
6/1/2021 • 2 minutes to read • [Edit Online](#)

Azure Maps has been available since May 2018. Azure Maps has been providing fresh map data, easy-to-use REST APIs, and powerful SDKs to support our enterprise customers with different kind of business use cases. The real world is changing every second, and it's crucial for us to provide a factual digital representation to our customers. Our customers that are planning to open or close facilities need our maps to update promptly. So, they can efficiently plan for delivery, maintenance, or customer service at the right facilities. We have created the Azure Maps data feedback site to empower our customers to provide direct data feedback. Customers' data feedback goes directly to our data providers and their map editors. They can quickly evaluate and incorporate feedback into our mapping products.

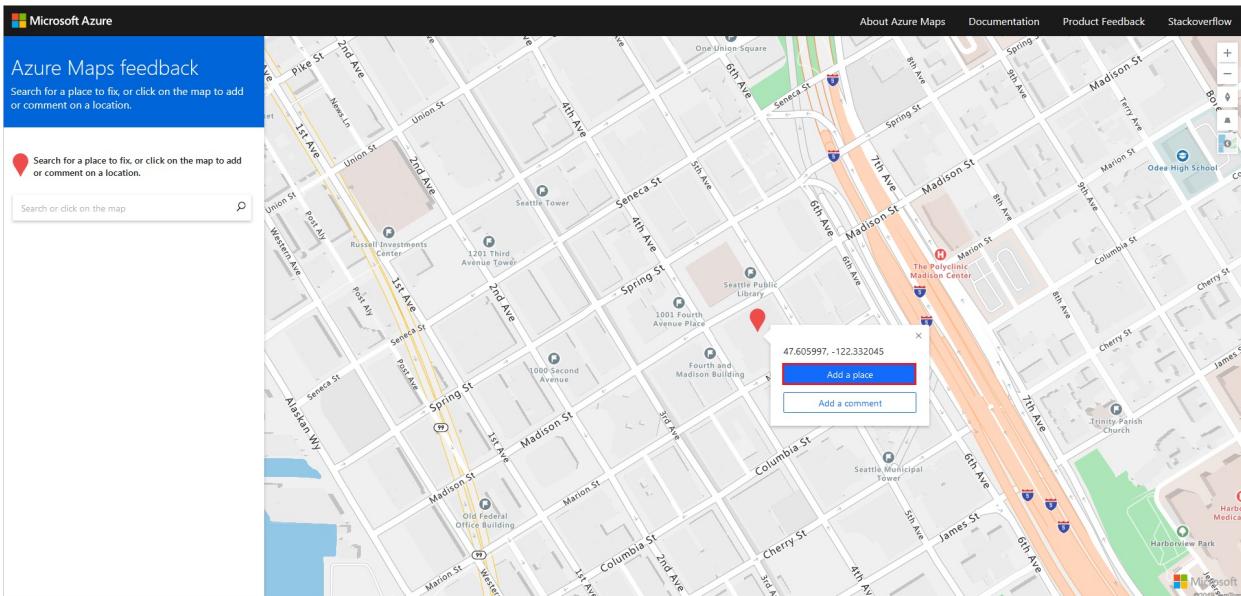
[Azure Maps Data feedback site](#) provides an easy way for our customers to provide map data feedback, especially on business points of interest and residential addresses. This article guides you on how to provide different kinds of feedback using the Azure Maps feedback site.

Add a business place or a residential address

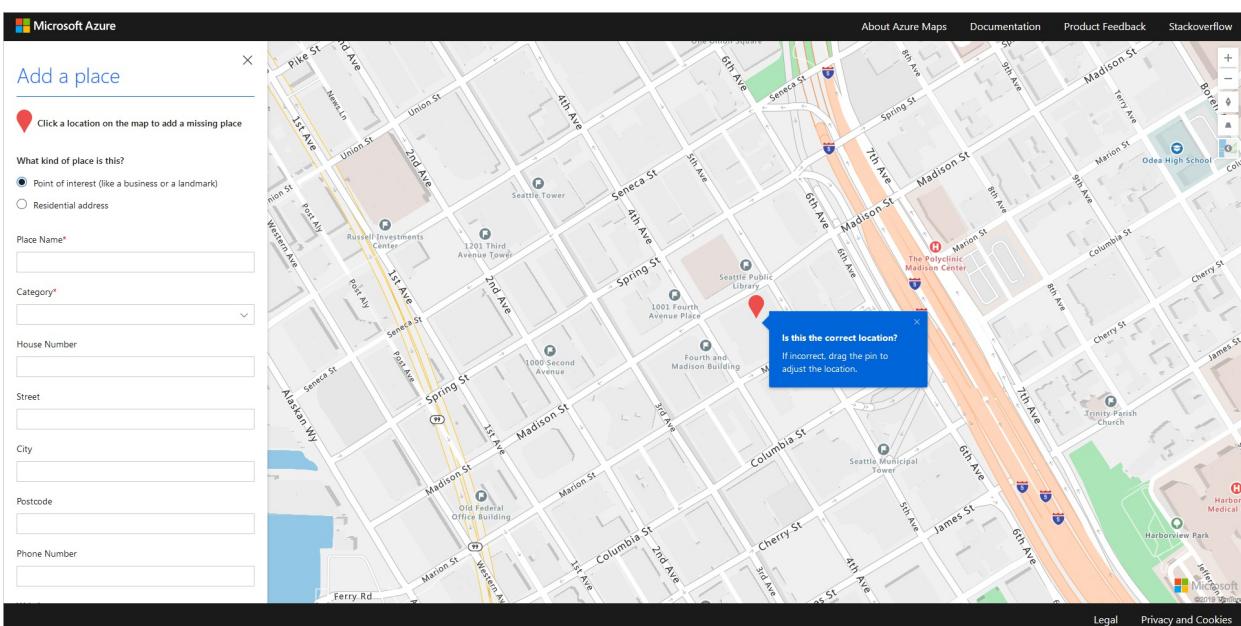
You may want to provide feedback about a missing point of interest or a residential address. There are two ways to do so. Open the Azure Map data feedback site, search for the missing location's coordinates, and then click "Add a place"



Or, you can interact with the map. Click on the location to drop a pin at the coordinate and click "Add a place".

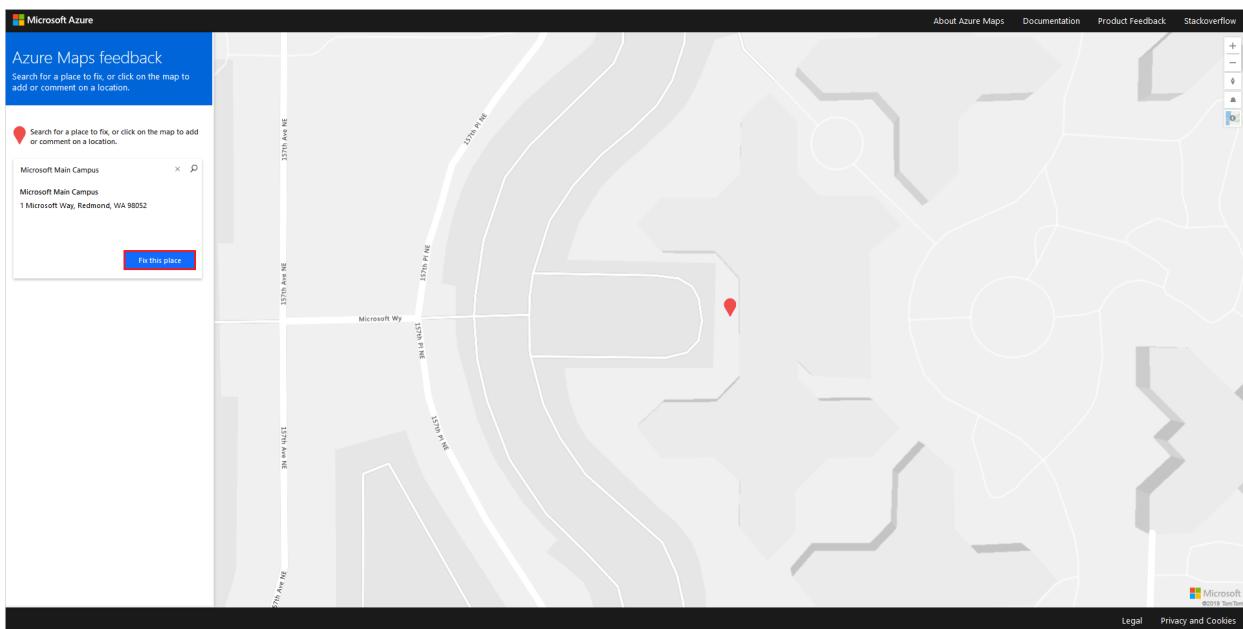


Upon clicking, you'll be directed to a form to provide the corresponding details for the place.



Fix a business place or a residential address

The feedback site also allows you to search and locate a business place or an address. You can provide feedback to fix the address or the pin location, if they aren't correct. To provide feedback to fix the address, use the search bar to search for a business place or residential address. Click on the location of your interest from the results list. Click on "Fix this place".



To provide feedback to fix the address, fill out the "Fix a place" form, and then click on the "submit" button.

Fix a place

This place does not exist

The pin location is incorrect

Place Name

Category

House Number

Street

City

Postcode

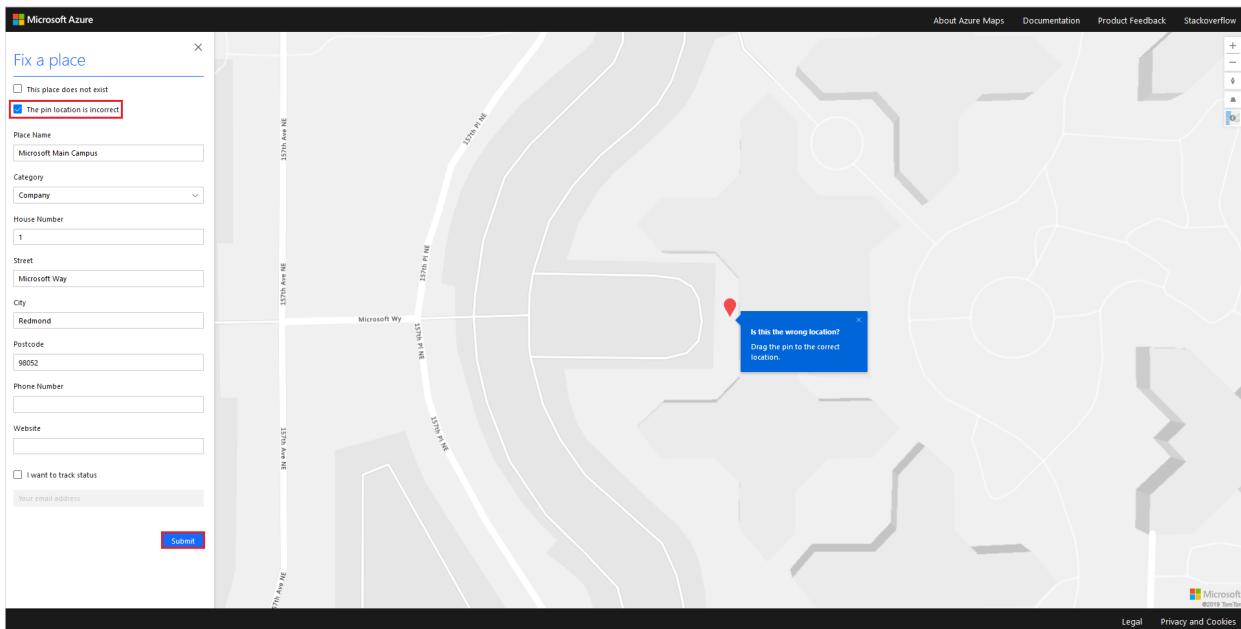
Phone Number

Website

I want to track status

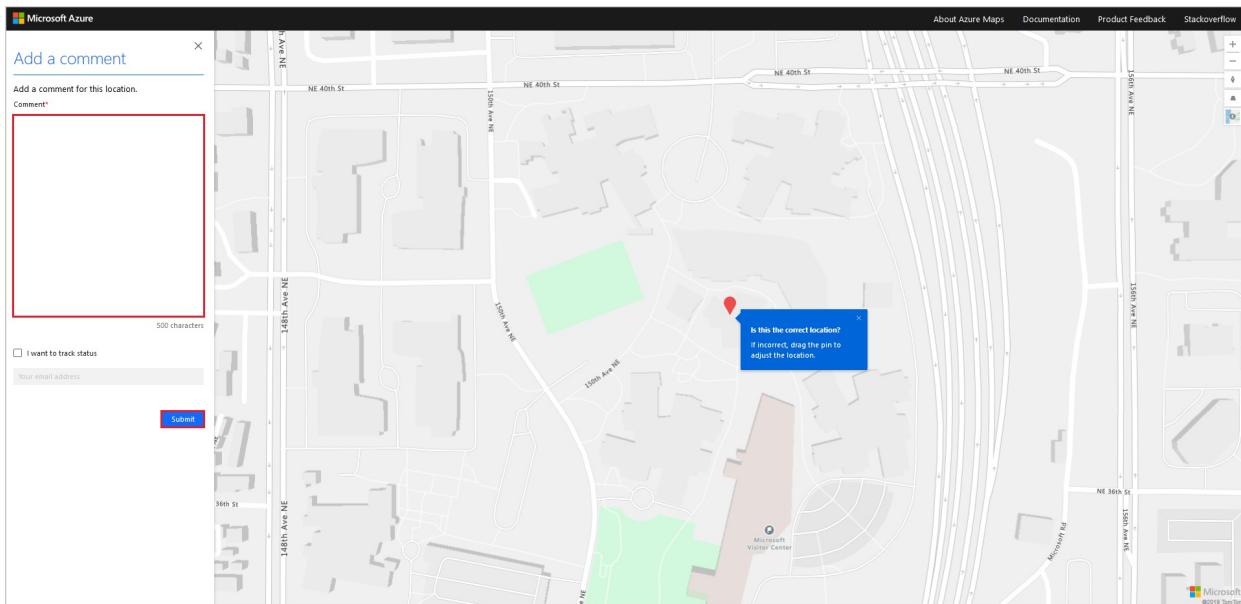
Your email address

If the pin location for the place is wrong, check the checkbox on the "Fix a place" form that says "The pin location is incorrect". Move the pin to the correct location, and then click the "submit" button.



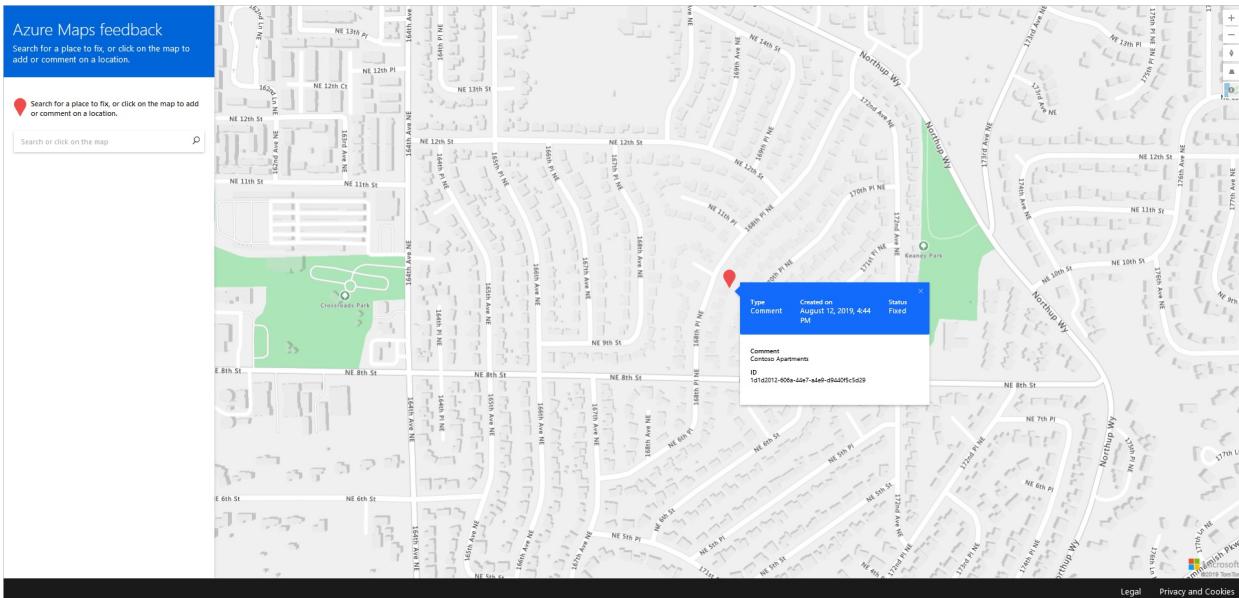
Add a comment

In addition to letting you search for a location, the feedback tool also lets you add a free form text comment for details related to the location. To add a comment, search for the location or click on the location. Click "Add a comment", write a comment, and then click "Submit".



Track status

You can also track the status of your request by checking the "I want to track status" box and providing your email while making a request. You will receive a tracking link in the email that provides an up-to-date status of your request.



Next steps

To post any technical questions related to Azure Maps, visit:

- Microsoft Q & A

Drawing package requirements

6/1/2021 • 13 minutes to read • [Edit Online](#)

You can convert uploaded Drawing packages into map data by using the [Azure Maps Conversion service](#). This article describes the Drawing package requirements for the Conversion API. To view a sample package, you can download the sample [Drawing package](#).

For a guide on how to prepare your Drawing package, see [Conversion Drawing Package Guide](#).

Prerequisites

The Drawing package includes drawings saved in DWG format, which is the native file format for Autodesk's AutoCAD® software.

You can choose any CAD software to produce the drawings in the Drawing package.

The [Azure Maps Conversion service](#) converts the Drawing package into map data. The Conversion service works with the AutoCAD DWG file format [AC1032](#).

Glossary of terms

For easy reference, here are some terms and definitions that are important as you read this article.

TERM	DEFINITION
Layer	An AutoCAD DWG layer from the drawing file.
Entity	An AutoCAD DWG entity from the drawing file.
Xref	A file in AutoCAD DWG file format, attached to the primary drawing as an external reference.
Level	An area of a building at a set elevation. For example, the floor of a building.
Feature	An instance of an object produced from the Conversion service that combines a geometry with metadata information.
Feature classes	A common blueprint for features. For example, a <i>unit</i> is a feature class, and an <i>office</i> is a feature.

Drawing package structure

A Drawing package is a .zip archive that contains the following files:

- DWG files in AutoCAD DWG file format.
- A *manifest.json* file that describes the DWG files in the Drawing package.

The Drawing package must be zipped into a single archive file, with the .zip extension. The DWG files can be organized in any way inside the package, but the manifest file must live at the root directory of the zipped package. The next sections detail the requirements for the DWG files, manifest file, and the content of these files.

To view a sample package, you can download the [sample Drawing package](#).

DWG file conversion process

The [Azure Maps Conversion service](#) performs the following on each DWG file:

- Extracts feature classes:
 - Levels
 - Units
 - Zones
 - Openings
 - Walls
 - Vertical penetrations
- Produces a *Facility* feature.
- Produces a minimal set of default Category features to be referenced by other features:
 - room
 - structure
 - wall
 - opening.door
 - zone
 - facility

DWG file requirements

A single DWG file is required for each level of the facility. All data of a single level must be contained in a single DWG file. Any external references (*xrefs*) must be bound to the parent drawing. For example, a facility with three levels will have three DWG files in the Drawing package.

Each DWG file must adhere to the following requirements:

- The DWG file must define the *Exterior* and *Unit* layers. It can optionally define the following layers: *Wall*, *Door*, *UnitLabel*, *Zone*, and *ZoneLabel*.
- The DWG file cannot contain features from multiple levels.
- The DWG file cannot contain features from multiple facilities.
- The DWG must reference the same measurement system and unit of measurement as other DWG files in the Drawing package.

DWG layer requirements

Each DWG layer must adhere to the following rules:

- A layer must exclusively contain features of a single class. For example, units and walls can't be in the same layer.
- A single class of features can be represented by multiple layers.
- Self-intersecting polygons are permitted, but are automatically repaired. When this occurs, the [Azure Maps Conversion service](#) raises a warning. It's advisable to manually inspect the repaired results, because they might not match the expected results.
- Each layer has a supported list of entity types. Any other entity types in a layer will be ignored. For example, text entities are not supported on the wall layer.

The table below outlines the supported entity types and converted map features for each layer. If a layer contains unsupported entity types, then the [Azure Maps Conversion service](#) ignores those entities.

LAYER	ENTITY TYPES	CONVERTED FEATURES
Exterior	Polygon, PolyLine (closed), Circle, Ellipse (closed)	Levels
Unit	Polygon, PolyLine (closed), Circle, Ellipse (closed)	Unit and Vertical penetrations
Wall	Polygon, PolyLine (closed), Circle, Ellipse (closed)	
Door	Polygon, PolyLine, Line, CircularArc, Circle	Openings
Zone	Polygon, PolyLine (closed), Circle, Ellipse (closed)	Zone
UnitLabel	Text (single line)	Not applicable. This layer can only add properties to the unit features from the Units layer. For more information, see the UnitLabel layer .
ZoneLabel	Text (single line)	Not applicable. This layer can only add properties to zone features from the ZonesLayer. For more information, see the ZoneLabel layer .

The sections below describe the requirements for each layer.

Exterior layer

The DWG file for each level must contain a layer to define that level's perimeter. This layer is referred to as the *exterior* layer. For example, if a facility contains two levels, then it needs to have two DWG files, with an exterior layer for each file.

No matter how many entity drawings are in the exterior layer, the [resulting facility dataset](#) will contain only one level feature for each DWG file. Additionally:

- Exteriors must be drawn as Polygon, PolyLine (closed), Circle, or Ellipse (closed).
- Exteriors may overlap, but are dissolved into one geometry.
- Resulting level feature must be at least 4 square meters.
- Resulting level feature must not be greater 400,000 square meters.

If the layer contains multiple overlapping PolyLines, the PolyLines are dissolved into a single Level feature.

Alternatively, if the layer contains multiple non-overlapping PolyLines, the resulting Level feature has a multi-polygonal representation.

You can see an example of the Exterior layer as the outline layer in the [sample Drawing package](#).

Unit layer

The DWG file for each level defines a layer containing units. Units are navigable spaces in the building, such as offices, hallways, stairs, and elevators. If the `VerticalPenetrationCategory` property is defined, navigable units that span multiple levels, such as elevators and stairs, are converted to Vertical Penetration features. Vertical penetration features that overlap each other are assigned one `setid`.

The Units layer should adhere to the following requirements:

- Units must be drawn as Polygon, PolyLine (closed), Circle, or Ellipse (closed).

- Units must fall inside the bounds of the facility exterior perimeter.
- Units must not partially overlap.
- Units must not contain any self-intersecting geometry.

Name a unit by creating a text object in the UnitLabel layer, and then place the object inside the bounds of the unit. For more information, see the [UnitLabel layer](#).

You can see an example of the Units layer in the [sample Drawing package](#).

Wall layer

The DWG file for each level can contain a layer that defines the physical extents of walls, columns, and other building structure.

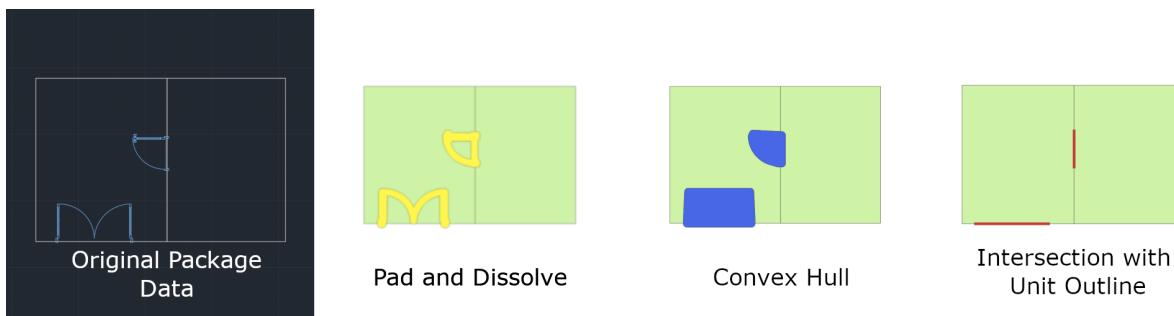
- Walls must be drawn as Polygon, PolyLine (closed), Circle, or Ellipse (closed).
- The wall layer or layers should only contain geometry that's interpreted as building structure.

You can see an example of the Walls layer in the [sample Drawing package](#).

Door layer

You can include a DWG layer that contains doors. Each door must overlap the edge of a unit from the Unit layer.

Door openings in an Azure Maps dataset are represented as a single-line segment that overlaps multiple unit boundaries. The following images show how to convert geometry in the Door layer to opening features in a dataset.



Zone layer

The DWG file for each level can contain a Zone layer that defines the physical extents of zones. A zone is a non-navigable space that can be named and rendered. Zones can span multiple levels and are grouped together using the zoneSetId property.

- Zones must be drawn as Polygon, PolyLine (closed), or Ellipse (closed).
- Zones can overlap.
- Zones can fall inside or outside the facility's exterior perimeter.

Name a zone by creating a text object in the ZoneLabel layer, and placing the text object inside the bounds of the zone. For more information, see [ZoneLabel layer](#).

You can see an example of the Zone layer in the [sample Drawing package](#).

UnitLabel layer

The DWG file for each level can contain a UnitLabel layer. The UnitLabel layer adds a name property to units extracted from the Unit layer. Units with a name property can have more details specified in the manifest file.

- Unit labels must be single-line text entities.
- Unit labels must fall entirely inside the bounds of their unit.
- Units must not contain multiple text entities in the UnitLabel layer.

You can see an example of the UnitLabel layer in the [sample Drawing package](#).

ZoneLabel layer

The DWG file for each level can contain a ZoneLabel layer. This layer adds a name property to zones extracted from the Zone layer. Zones with a name property can have more details specified in the manifest file.

- Zones labels must be single-line text entities.
- Zones labels must fall inside the bounds of their zone.
- Zones must not contain multiple text entities in the ZoneLabel layer.

You can see an example of the ZoneLabel layer in the [sample Drawing package](#).

Manifest file requirements

The zip folder must contain a manifest file at the root level of the directory, and the file must be named **manifest.json**. It describes the DWG files to allow the [Azure Maps Conversion service](#) to parse their content. Only the files identified by the manifest are ingested. Files that are in the zip folder, but aren't properly listed in the manifest, are ignored.

The file paths in the `buildingLevels` object of the manifest file must be relative to the root of the zip folder. The DWG file name must exactly match the name of the facility level. For example, a DWG file for the "Basement" level is "Basement.dwg." A DWG file for level 2 is named as "level_2.dwg." Use an underscore, if your level name has a space.

Although there are requirements when you use the manifest objects, not all objects are required. The following table shows the required and optional objects for version 1.1 of the [Azure Maps Conversion service](#).

NOTE

Unless otherwise specified, all properties with a string property type allow for one thousand characters.

OBJECT	REQUIRED	DESCRIPTION
<code>version</code>	true	Manifest schema version. Currently, only version 1.1 is supported.
<code>directoryInfo</code>	true	Outlines the facility geographic and contact information. It can also be used to outline an occupant geographic and contact information.
<code>buildingLevels</code>	true	Specifies the levels of the buildings and the files containing the design of the levels.
<code>georeference</code>	true	Contains numerical geographic information for the facility drawing.
<code>dwgLayers</code>	true	Lists the names of the layers, and each layer lists the names of its own features.
<code>unitProperties</code>	false	Can be used to insert more metadata for the unit features.
<code>zoneProperties</code>	false	Can be used to insert more metadata for the zone features.

The next sections detail the requirements for each object.

directoryInfo

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>name</code>	string	true	Name of building.
<code>streetAddress</code>	string	false	Address of building.
<code>unit</code>	string	false	Unit in building.
<code>locality</code>	string	false	Name of an city, town, area, neighborhood, or region.
<code>adminDivisions</code>	JSON array of strings	false	An array containing address designations. For example: (Country, State) Use ISO 3166 country codes and ISO 3166-2 state/territory codes.
<code>postalCode</code>	string	false	The mail sorting code.
<code>hoursOfOperation</code>	string	false	Adheres to the OSM Opening Hours format.
<code>phone</code>	string	false	Phone number associated with the building.
<code>website</code>	string	false	Website associated with the building.
<code>nonPublic</code>	bool	false	Flag specifying if the building is open to the public.
<code>anchorLatitude</code>	numeric	false	Latitude of a facility anchor (pushpin).
<code>anchorLongitude</code>	numeric	false	Longitude of a facility anchor (pushpin).
<code>anchorHeightAboveSeaLevel</code>	numeric	false	Height of the facility's ground floor above sea level, in meters.
<code>defaultLevelVerticalExtent</code>	numeric	false	Default height (thickness) of a level of this facility to use when a level's <code>verticalExtent</code> is undefined.

buildingLevels

The `buildingLevels` object contains a JSON array of buildings levels.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>levelName</code>	string	true	Descriptive level name. For example: Floor 1, Lobby, Blue Parking, or Basement.
<code>ordinal</code>	integer	true	Determines the vertical order of levels. Every facility must have a level with ordinal 0.
<code>heightAboveFacilityAnchor</code>	numeric	false	Level height above the anchor in meters.
<code>verticalExtent</code>	numeric	false	Floor-to-ceiling height (thickness) of the level in meters.
<code>filename</code>	string	true	File system path of the CAD drawing for a building level. It must be relative to the root of the building's zip file.

georeference

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>lat</code>	numeric	true	Decimal representation of degrees latitude at the facility drawing's origin. The origin coordinates must be in WGS84 Web Mercator (EPSG:3857).
<code>lon</code>	numeric	true	Decimal representation of degrees longitude at the facility drawing's origin. The origin coordinates must be in WGS84 Web Mercator (EPSG:3857).
<code>angle</code>	numeric	true	The clockwise angle, in degrees, between true north and the drawing's vertical (Y) axis.

dwgLayers

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>exterior</code>	array of strings	true	Names of layers that define the exterior building profile.
<code>unit</code>	array of strings	true	Names of layers that define units.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>wall</code>	array of strings	false	Names of layers that define walls.
<code>door</code>	array of strings	false	Names of layers that define doors.
<code>unitLabel</code>	array of strings	false	Names of layers that define names of units.
<code>zone</code>	array of strings	false	Names of layers that define zones.
<code>zoneLabel</code>	array of strings	false	Names of layers that define names of zones.

`unitProperties`

The `unitProperties` object contains a JSON array of unit properties.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>unitName</code>	string	true	Name of unit to associate with this <code>unitProperty</code> record. This record is only valid when a label matching <code>unitName</code> is found in the <code>unitLabel</code> layers.
<code>categoryName</code>	string	false	Purpose of the unit. A list of values that the provided rendering styles can make use of is available here .
<code>occupants</code>	array of directoryInfo objects	false	List of occupants for the unit.
<code>nameAlt</code>	string	false	Alternate name of the unit.
<code>nameSubtitle</code>	string	false	Subtitle of the unit.
<code>addressRoomNumber</code>	string	false	Room, unit, apartment, or suite number of the unit.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>verticalPenetrationCategory</code>	string	false	When this property is defined, the resulting feature is a vertical penetration (VRT) rather than a unit. You can use vertical penetrations to go to other vertical penetration features in the levels above or below it. Vertical penetration is a Category name. If this property is defined, the <code>categoryName</code> property is overridden with <code>verticalPenetrationCategory</code> .
<code>verticalPenetrationDirection</code>	string	false	If <code>verticalPenetrationCategory</code> is defined, optionally define the valid direction of travel. The permitted values are: <code>lowToHigh</code> , <code>highToLow</code> , <code>both</code> , and <code>closed</code> . The default value is <code>both</code> .
<code>nonPublic</code>	bool	false	Indicates if the unit is open to the public.
<code>isRoutable</code>	bool	false	When this property is set to <code>false</code> , you can't go to or through the unit. The default value is <code>true</code> .
<code>isOpenArea</code>	bool	false	Allows the navigating agent to enter the unit without the need for an opening attached to the unit. By default, this value is set to <code>true</code> for units with no openings, and <code>false</code> for units with openings. Manually setting <code>isOpenArea</code> to <code>false</code> on a unit with no openings results in a warning, because the resulting unit won't be reachable by a navigating agent.

`zoneProperties`

The `zoneProperties` object contains a JSON array of zone properties.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
----------	------	----------	-------------

PROPERTY	TYPE	REQUIRED	DESCRIPTION
zoneName	string	true	Name of zone to associate with <code>zoneProperty</code> record. This record is only valid when a label matching <code>zoneName</code> is found in the <code>zoneLabel</code> layer of the zone.
categoryName	string	false	Purpose of the unit. A list of values that the provided rendering styles can make use of is available here .
zoneNameAlt	string	false	Alternate name of the zone.
zoneNameSubtitle	string	false	Subtitle of the zone.
zoneSetId	string	false	Set ID to establish a relationship among multiple zones so that they can be queried or selected as a group. For example, zones that span multiple levels.

Sample Drawing package manifest

Below is the manifest file for the sample Drawing package. To download the entire package, see [sample Drawing package](#).

Manifest file

```
{
  "version": "1.1",
  "directoryInfo": {
    "name": "ContosoBuilding",
    "streetAddress": "Contoso Way",
    "unit": "1",
    "locality": "Contoso eastside",
    "postalCode": "98052",
    "adminDivisions": [
      "Contoso city",
      "Contoso state",
      "Contoso country"
    ],
    "hoursOfOperation": "Mo-Fr08:00-17:00open",
    "phone": "1(425)555-1234",
    "website": "www.contoso.com",
    "nonPublic": false,
    "anchorLatitude": 47.636152,
    "anchorLongitude": -122.132600,
    "anchorHeightAboveSeaLevel": 1000,
    "defaultLevelVerticalExtent": 3
  },
  "buildingLevels": {
    "levels": [
      {
        "levelName": "Basement",
        "ordinal": -1,
        "filename": "./Basement.dwg"
      },
      {
        "levelName": "Ground",
        "ordinal": 0,
        "filename": "./Ground.dwg"
      }
    ]
  }
}
```

```
"ordinal":0,
"verticalExtent":5,
"filename":"./Ground.dwg"
},{
"levelName":"Level2",
"ordinal":1,
"heightAboveFacilityAnchor":3.5,
"filename":"./Level_2.dwg"
}
]
},
"georeference":{
"lat":47.636152,
"lon":-122.132600,
"angle":0
},
"dwgLayers":{
"exterior":[
"OUTLINE","WINDOWS"
],
"unit":[
"UNITS"
],
"wall":[
"WALLS"
],
"door":[
"DOORS"
],
"unitLabel":[
"UNITLABELS"
],
"zone":[
"ZONES"
],
"zoneLabel":[
"ZONELABELS"
]
},
"unitProperties":[
{
"unitName":"B01",
"categoryName":"room.office",
"occupants":[
{
"name":"Joe's Office",
"phone":"1(425)555-1234"
}
],
"nameAlt":"Basement01",
"nameSubtitle":"01",
"addressRoomNumber":"B01",
"nonPublic":true,
"isRoutable":true,
"isOpenArea":true
},
{
"unitName":"B02"
},
{
"unitName":"B05",
"categoryName":"room.office"
},
{
"unitName":"STRB01",
"verticalPenetrationCategory":"verticalPenetration.stairs",
"verticalPenetrationDirection":"both"
},
{
```

```
"unitName":"ELVB01",
"verticalPenetrationCategory":"verticalPenetration.elevator",
"verticalPenetrationDirection":"high_to_low"
},
],
"zoneProperties":
[
{
"zoneName":"WifiB01",
"categoryName":"Zone",
"zoneNameAlt":"MyZone",
"zoneNameSubtitle":"Wifi",
"zoneSetId":"1234"
},
{
"zoneName":"Wifi101",
"categoryName":"Zone",
"zoneNameAlt":"MyZone",
"zoneNameSubtitle":"Wifi",
"zoneSetId":"1234"
}
]
```

Next steps

When your Drawing package meets the requirements, you can use the [Azure Maps Conversion service](#) to convert the package to a map dataset. Then, you can use the dataset to generate an indoor map by using the indoor maps module.

[Creator Facility Ontology](#)

[Creator for indoor maps](#)

[Drawing Package Guide](#)

[Creator for indoor maps](#)

[Tutorial: Creating a Creator indoor map](#)

[Indoor maps dynamic styling](#)

Conversion Drawing package guide

6/1/2021 • 9 minutes to read • [Edit Online](#)

This guide shows you how to prepare your Drawing Package for the [Azure Maps Conversion service](#) using specific CAD commands to correctly prepare your DWG files and manifest file for the Conversion service.

To start with, make sure your Drawing Package is in .zip format, and contains the following files:

- One or more drawing files in DWG format.
- A Manifest file describing DWG files and facility metadata.

If you don't have your own package to reference along with this guide, you may download the [sample Drawing package](#).

You may choose any CAD software to open and prepare your facility drawing files. However, this guide is created using Autodesk's AutoCAD® software. Any commands referenced in this guide are meant to be executed using Autodesk's AutoCAD® software.

TIP

For more information about drawing package requirements that aren't covered in this guide, see [Drawing Package Requirements](#).

Glossary of terms

For easy reference, here are some terms and definitions that are important as you read this guide.

TERM	DEFINITION
Layer	An AutoCAD DWG layer from the drawing file.
Entity	An AutoCAD DWG entity from the drawing file.
Level	An area of a building at a set elevation. For example, the floor of a building.
Feature	An object that combines a geometry with more metadata information.
Feature classes	A common blueprint for features. For example, a <i>unit</i> is a feature class, and an <i>office</i> is a feature.

Step 1: DWG file requirements

When preparing your facility drawing files for the Conversion service, make sure to follow these preliminary requirements and recommendations:

- Facility drawing files must be saved in DWG format, which is the native file format for Autodesk's AutoCAD® software.
- The Conversion service works with the AutoCAD DWG file format.AC1032is the internal format version

for the DWG files, and it's a good idea to select AC1032 for the internal DWG file format version.

- A DWG file can only contain a single floor. A floor of a facility must be provided in its own separate DWG file. So, if you have five floors in a facility, you must create five separate DWG files.

Step 2: Prepare the DWG files

This part of the guide will show you how to use CAD commands to ensure that your DWG files meet the requirements of the Conversion service.

You may choose any CAD software to open and prepare your facility drawing files. However, this guide is created using Autodesk's AutoCAD® software. Any commands referenced in this guide are meant to be executed using Autodesk's AutoCAD® software.

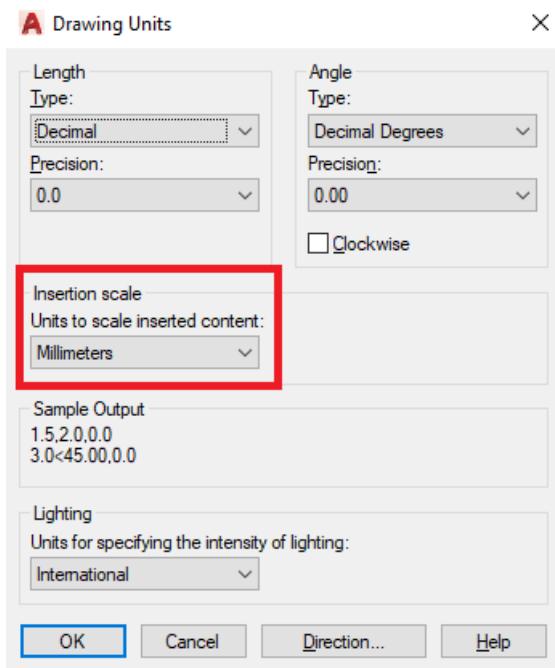
Bind External References

Each floor of a facility must be provided as one DWG file. If there are no external references, then nothing more needs to be done. However, if there are any external references, they must be bound to a single drawing. To bind an external reference, you may use the `XREF` command. After binding, each external reference drawing will be added as a block reference. If you need to make changes to any of these layers, remember to explode the block references by using the `Xplode` command.

Unit of measurement

The drawings can be created using any unit of measurement. However, all drawings must use the same unit of measurement. So, if one floor of the facility is using millimeters, then all other floors (drawings) must also be in millimeters. You can verify or modify the measurement unit by using the `UNITS` command.

The following image shows the Drawing Units window within Autodesk's AutoCAD® software that you can use to verify the unit of measurement.



Alignment

Each floor of a facility is provided as an individual DWG file. As a result, it's possible that the floors are not perfectly aligned when stacked on top of each other. Azure Maps Conversion service requires that all drawings be aligned with the physical space. To verify alignment, use a reference point that can span across floors, such as an elevator or column that spans multiple floors. You can view all the floors by opening a new drawing, and then use the `XATTACH` command to load all floor drawings. If you need to fix any alignment issues, you can use the `MOVE` command to realign the floors that require it.

Layers

Ensure that each layer of a drawing contains entities of one feature class. If a layer contains entities for walls, then it can't have other features such as units or doors. However, a feature class can be split up over multiple layers. For example, you can have three layers in the drawing that contain wall entities.

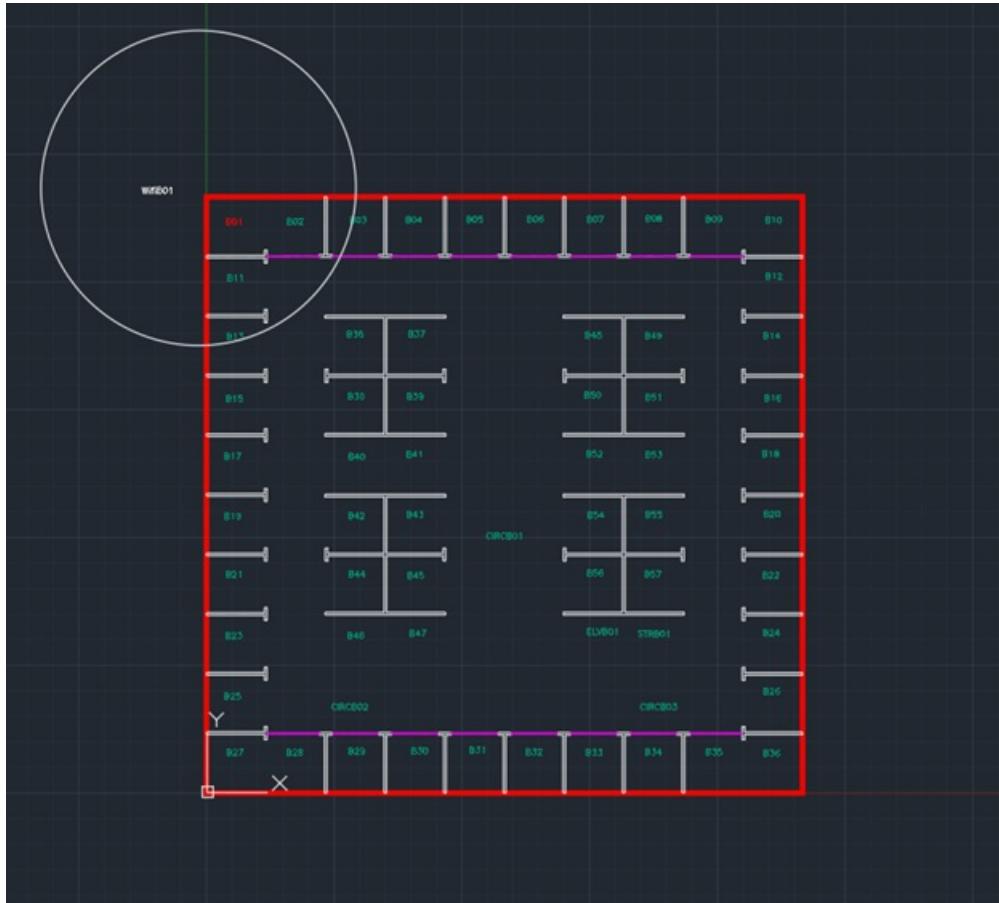
Furthermore, each layer has a list of supported entity types and any other types are ignored. For example, if the Unit Label layer only supports single-line text, a multiline text or Polyline on the same layer is ignored.

For a better understanding of layers and feature classes, see [Drawing Package Requirements](#).

Exterior layer

A single level feature is created from each exterior layer or layers. This level feature defines the level's perimeter. It's important to ensure that the entities in the exterior layer meet the requirements of the layer. For example, a closed Polyline is supported; but an open Polyline isn't. If your exterior layer is made of multiple line segments, they must be provided as one closed Polyline. To join multiple line segments together, select all line segments and use the `JOIN` command.

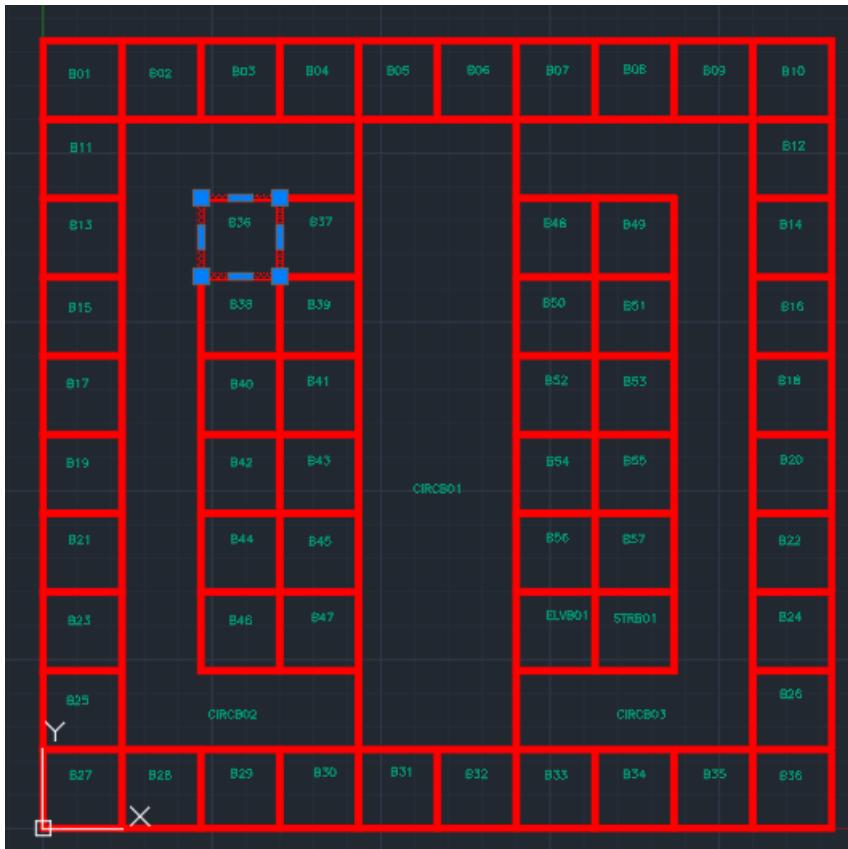
The following image is taken from the sample package, and shows the exterior layer of the facility in red. The unit layer is turned off to help with visualization.



Unit layer

Units are navigable spaces in the building, such as offices, hallways, stairs, and elevators. A closed entity type such as Polygon, closed Polyline, Circle, or closed Ellipse is required to represent each unit. So, walls and doors alone won't create a unit because there isn't an entity that represents the unit.

The following image is taken from the [sample Drawing package](#) and shows the unit label layer and unit layer in red. All other layers are turned off to help with visualization. Also, one unit is selected to help show that each unit is a closed Polyline.



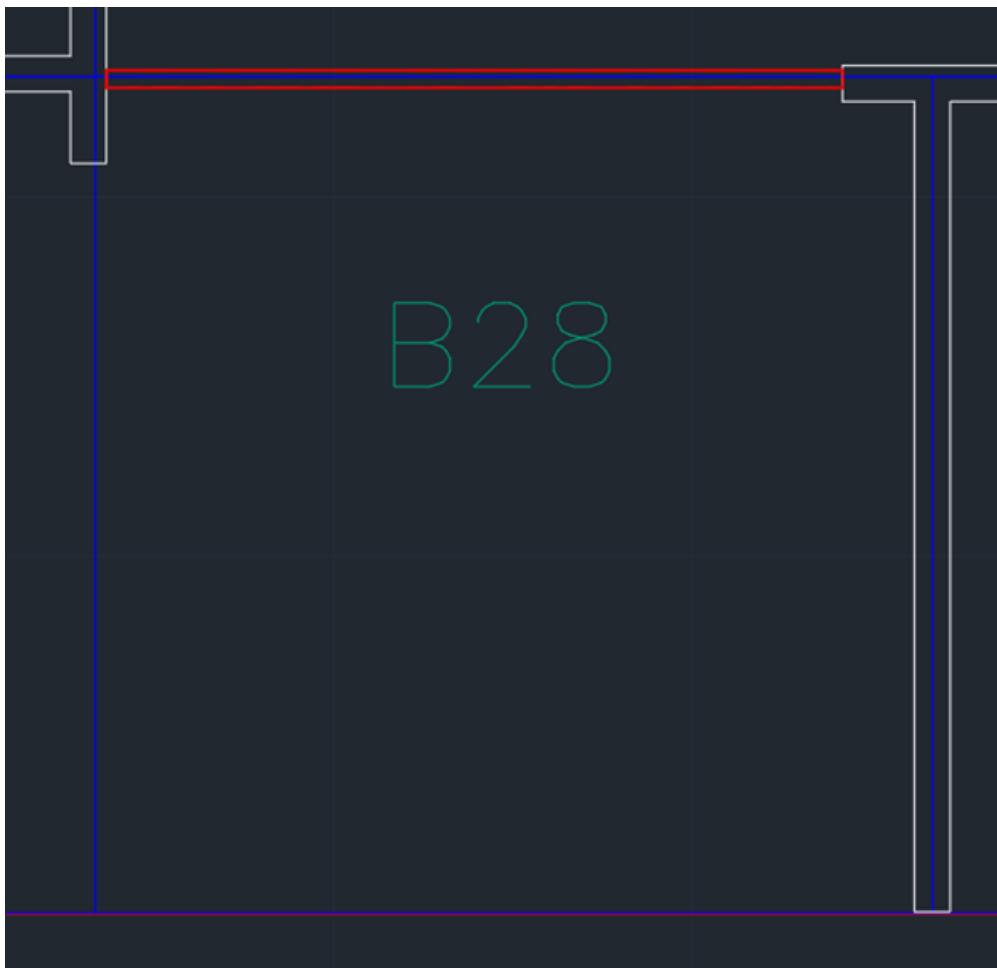
Unit label layer

If you'd like to add a name property to a unit, you'll need to add a separate layer for unit labels. Labels must be provided as single-line text entities that fall inside the bounds of a unit. A corresponding unit property must be added to the manifest file where the `unitName` matches the Contents of the Text. To learn about all supported unit properties, see [unitProperties](#).

Door layer

Doors are optional. However, doors may be used if you'd like to specify the entry point(s) for a unit. Doors can be drawn in any way if it's a supported entity type by the door layer. The door must overlap the boundary of a unit and the overlapping edge of the unit is then be treated as an opening to the unit.

The following image is taken from the [sample Drawing package](#) and shows a unit with a door (in red) drawn on the unit boundary.



Wall layer

The wall layer is meant to represent the physical extents of a facility such as walls and columns. The Azure Maps Conversion service perceives walls as physical structures that are an obstruction to routing. With that in mind, a wall should be thought as a physical structure that one can see, but not walk though. Anything that can't be seen won't be captured in this layer. If a wall has inner walls or columns inside, then only the exterior should be captured.

Step 3: Prepare the manifest

The Drawing package Manifest is a JSON file. The Manifest tells the Azure Maps Conversion service how to read the facility DWG files and metadata. Some examples of this information could be the specific information each DWG layer contains, or the geographical location of the facility.

To achieve a successful conversion, all "required" properties must be defined. A sample manifest file can be found inside the [sample Drawing package](#). This guide does not cover properties supported by the manifest. For more information about manifest properties, see [Manifest File Properties](#).

Building levels

The building level specifies which DWG file to use for which level. A level must have a level name and ordinal that describes that vertical order of each level. Every facility must have an ordinal 0, which is the ground floor of a facility. An ordinal 0 must be provided even if the drawings occupy a few floors of a facility. For example, floors 15-17 can be defined as ordinal 0-2, respectively.

The following example is taken from the [sample Drawing package](#). The facility has three levels: basement, ground, and level 2. The filename contains the full file name and path of the file relative to the manifest file within the .zip Drawing package.

```

"buildingLevels": {
    "levels": [
        {
            "levelName": "Basement",
            "ordinal": -1,
            "filename": "./Basement.dwg"
        },
        {
            "levelName": "Ground",
            "ordinal": 0,
            "filename": "./Ground.dwg"
        },
        {
            "levelName": "Level2",
            "ordinal": 1,
            "filename": "./Level_2.dwg"
        }
    ]
},

```

georeference

The `georeference` object is used to specify where the facility is located geographically and how much to rotate the facility. The origin point of the drawing should match the latitude and longitude provided with the `georeference` object. The clockwise angle, in degrees, between true north and the drawing's vertical (Y) axis.

dwgLayers

The `dwgLayers` object is used to specify that DWG layer names where feature classes can be found. To receive a property converted facility, it's important to provide the correct layer names. For example, a DWG wall layer must be provided as a wall layer and not as a unit layer. The drawing can have other layers such as furniture or plumbing; but, they'll be ignored by the Azure Maps Conversion service if they're not specified in the manifest.

The following example of the `dwgLayers` object in the manifest.

```

"dwgLayers": {
    "exterior": [
        "OUTLINE"
    ],
    "unit": [
        "UNITS"
    ],
    "wall": [
        "WALLS"
    ],
    "door": [
        "DOORS"
    ],
    "unitLabel": [
        "UNITLABELS"
    ],
    "zone": [
        "ZONES"
    ],
    "zoneLabel": [
        "ZONELABELS"
    ]
}

```

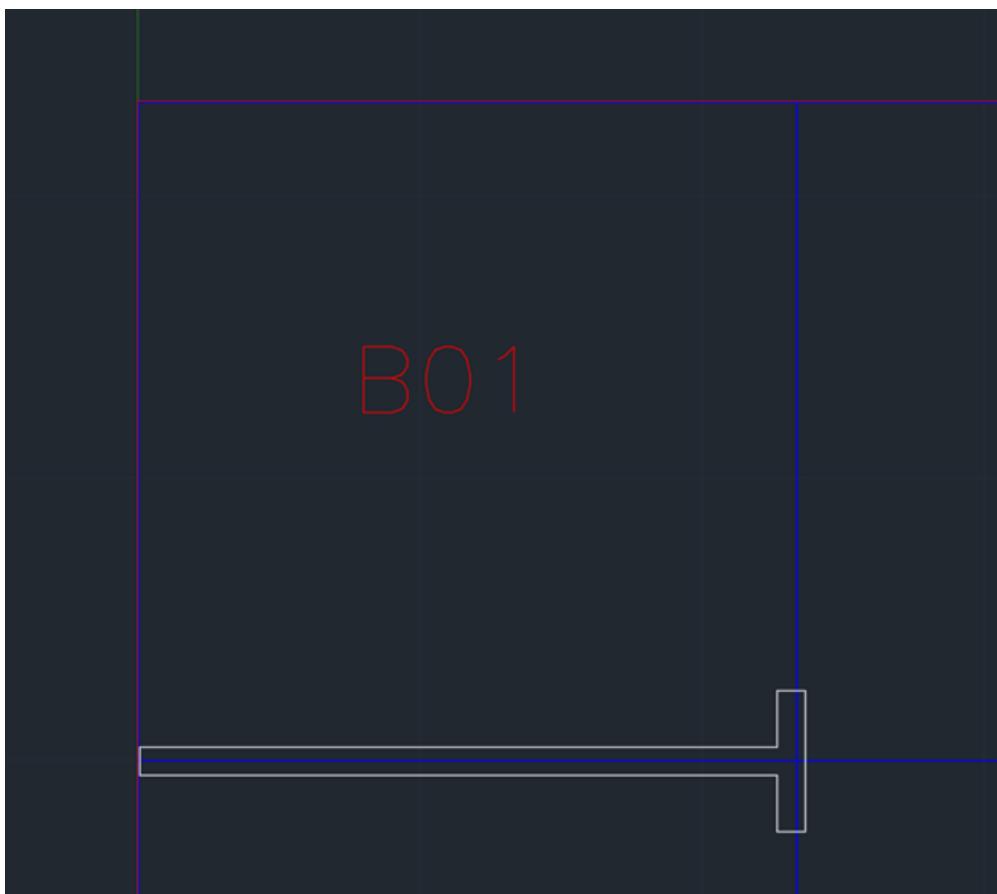
The following image shows the layers from the corresponding DWG drawing viewed in Autodesk's AutoCAD® software.

Status	Name
✓	0
✗	A-Anno-Scrn
✗	A-Wall
✗	Defpoints
✗	DOORS
✗	G-Anno-Nplt
✗	LABELS
✗	OUTLINE
✗	UNITS
✗	WALLS

unitProperties

The `unitProperties` object allows you to define other properties for a unit that you can't do in the DWG file. Examples could be directory information of a unit or the category type of a unit. A unit property is associated with a unit by having the `unitName` object match the label in the `unitLabel` layer.

The following image is taken from the [sample Drawing package](#). It displays the unit label that's associated to the unit property in the manifest.



The following snippet shows the unit property object that is associated with the unit.

```
"unitProperties": [
  {
    "unitName": "B01",
    "categoryName": "room.office",
    "navigableBy": ["pedestrian", "wheelchair", "machine"],
    "routeThroughBehavior": "disallowed",
    "occupants": [
      {
        "name": "Joe's Office",
        "phone": "1(425)555-1234"
      }
    ],
    "nameAlt": "Basement01",
    "nameSubtitle": "01",
    "addressRoomNumber": "B01",
    "nonPublic": true,
    "isRoutable": true,
    "isOpenArea": true
  },
]
```

Step 4: Prepare the Drawing Package

You should now have all the DWG drawings prepared to meet Azure Maps Conversion service requirements. A manifest file has also been created to help describe the facility. All files will need to be zipped into a single archive file, with the `.zip` extension. It's important that the manifest file is named `manifest.json` and is placed in the root directory of the zipped package. All other files can be in any directory of the zipped package if the filename includes the relative path to the manifest. For an example of a drawing package, see the [sample Drawing package](#).

Next steps

[Tutorial: Creating a Creator indoor map](#)

StylesObject Schema reference guide for dynamic Maps

6/1/2021 • 4 minutes to read • [Edit Online](#)

The `StylesObject` is a `StyleObject` array representing stateset styles. Use the Azure Maps Creator [Feature State service](#) to apply your stateset styles to indoor map data features. Once you've created your stateset styles and associated them with indoor map features, you can use them to create dynamic indoor maps. For more information on creating dynamic indoor maps, see [Implement dynamic styling for Creator indoor maps](#).

StyleObject

A `StyleObject` is one of the following style rules:

- [BooleanTypeStyleRule](#)
- [NumericTypeStyleRule](#)
- [StringTypeStyleRule](#)

The JSON below shows example usage of each of the three style types. The `BooleanTypeStyleRule` is used to determine the dynamic style for features whose `occupied` property is true and false. The `NumericTypeStyleRule` is used to determine the style for features whose `temperature` property falls within a certain range. Finally, the `StringTypeStyleRule` is used to match specific styles to `meetingType`.

```

"styles": [
  {
    "keyname": "occupied",
    "type": "boolean",
    "rules": [
      {
        "true": "#FF0000",
        "false": "#00FF00"
      }
    ]
  },
  {
    "keyname": "temperature",
    "type": "number",
    "rules": [
      {
        "range": {
          "minimum": 50,
          "exclusiveMaximum": 70
        },
        "color": "#343deb"
      },
      {
        "range": {
          "maximum": 70,
          "exclusiveMinimum": 30
        },
        "color": "#eba834"
      }
    ]
  },
  {
    "keyname": "meetingType",
    "type": "string",
    "rules": [
      {
        "private": "#FF0000",
        "confidential": "#FF00AA",
        "allHands": "#00FF00",
        "brownBag": "#964B00"
      }
    ]
  }
]

```

NumericTypeStyleRule

A `NumericTypeStyleRule` is a `StyleObject` and consists of the following properties:

PROPERTY	TYPE	DESCRIPTION	REQUIRED
<code>keyName</code>	string	The <i>state</i> or dynamic property name. A <code>keyName</code> should be unique inside the <code>StyleObject</code> array.	Yes
<code>type</code>	string	Value is "numeric".	Yes

PROPERTY	TYPE	DESCRIPTION	REQUIRED
rules	NumberRuleObject []	An array of numeric style ranges with associated colors. Each range defines a color that's to be used when the <i>state</i> value satisfies the range.	Yes

NumberRuleObject

A NumberRuleObject consists of a RangeObject and a color property. If the state value falls into the range, its color for display will be the color specified in the color property.

If you define multiple overlapping ranges, the color chosen will be the color that's defined in the first range that is satisfied.

In the following JSON sample, both ranges will hold true when the state value is between 50-60. However, the color that will be used is #343deb because it's the first range in the list that has been satisfied.

```
{
  "rules": [
    {
      "range": {
        "minimum": 50,
        "exclusiveMaximum": 70
      },
      "color": "#343deb"
    },
    {
      "range": {
        "minimum": 50,
        "maximum": 60
      },
      "color": "#eba834"
    }
  ]
}
```

PROPERTY	TYPE	DESCRIPTION	REQUIRED
range	RangeObject	The RangeObject defines a set of logical range conditions, which, if true, change the display color of the state to the color specified in the color property. If range is unspecified, then the color defined in the color property will always be used.	No

PROPERTY	TYPE	DESCRIPTION	REQUIRED
<code>color</code>	string	The color to use when state value falls into the range. The <code>color</code> property is a JSON string in any one of following formats: <ul style="list-style-type: none"> • HTML-style hex values • RGB ("#ff00", "#ffff00", "rgb(255, 255, 0)") • RGBA ("rgba(255, 255, 0, 1)") • HSL("hsl(100, 50%, 50%)") • HSLA("hsla(100, 50%, 50%, 1)") • Predefined HTML colors names, like yellow, and blue. 	Yes

RangeObject

The `RangeObject` defines a numeric range value of a `NumberRuleObject`. For the `state` value to fall into the range, all defined conditions must hold true.

PROPERTY	TYPE	DESCRIPTION	REQUIRED
<code>minimum</code>	double	All the number x that $x \geq \text{minimum}$.	No
<code>maximum</code>	double	All the number x that $x \leq \text{maximum}$.	No
<code>exclusiveMinimum</code>	double	All the number x that $x > \text{exclusiveMinimum}$.	No
<code>exclusiveMaximum</code>	double	All the number x that $x < \text{exclusiveMaximum}$.	No

Example of NumericTypeStyleRule

The following JSON illustrates a `NumericTypeStyleRule` `state` named `temperature`. In this example, the `NumberRuleObject` contains two defined temperature ranges and their associated color styles. If the temperature range is 50-69, the display should use the color `#343deb`. If the temperature range is 31-70, the display should use the color `#eba834`.

```
{
  "keyname": "temperature",
  "type": "number",
  "rules": [
    {
      "range": {
        "minimum": 50,
        "exclusiveMaximum": 70
      },
      "color": "#343deb"
    },
    {
      "range": {
        "maximum": 70,
        "exclusiveMinimum": 30
      },
      "color": "#eba834"
    }
  ]
}
```

StringTypeStyleRule

A `StringTypeStyleRule` is a `StyleObject` and consists of the following properties:

PROPERTY	TYPE	DESCRIPTION	REQUIRED
<code>keyName</code>	string	The <i>state</i> or dynamic property name. A <code>keyName</code> should be unique inside the <code>StyleObject</code> array.	Yes
<code>type</code>	string	Value is "string".	Yes
<code>rules</code>	<code>StringRuleObject</code> []	An array of N number of <i>state</i> values.	Yes

StringRuleObject

A `StringRuleObject` consists of up to N number of state values that are the possible string values of a feature's property. If the feature's property value doesn't match any of the defined state values, that feature won't have a dynamic style. If duplicate state values are given, the first one takes precedence.

The string value matching is case-sensitive.

PROPERTY	TYPE	DESCRIPTION	REQUIRED
<code>stateValue1</code>	string	The color when value string is <code>stateValue1</code> .	No
<code>stateValue2</code>	string	The color when value string is <code>stateValue2</code> .	No
<code>stateValueN</code>	string	The color when value string is <code>stateValueN</code> .	No

Example of StringTypeStyleRule

The following JSON illustrates a `StringStyleRule` that defines styles associated with specific meeting types.

```
{  
  "keyname": "meetingType",  
  "type": "string",  
  "rules": [  
    {  
      "private": "#FF0000",  
      "confidential": "#FF00AA",  
      "allHands": "#00FF00",  
      "brownBag": "#964B00"  
    }  
  ]  
}
```

BooleanTypeStyleRule

A `BooleanTypeStyleRule` is a `StyleObject` and consists of the following properties:

PROPERTY	TYPE	DESCRIPTION	REQUIRED
<code>keyName</code>	string	The <i>state</i> or dynamic property name. A <code>keyName</code> should be unique inside the <code>StyleObject</code> array.	Yes
<code>type</code>	string	Value is "boolean".	Yes
<code>rules</code>	<code>BooleanRuleObject</code> [1]	A boolean pair with colors for <code>true</code> and <code>false</code> <i>state</i> values.	Yes

BooleanRuleObject

A `BooleanRuleObject` defines colors for `true` and `false` values.

PROPERTY	TYPE	DESCRIPTION	REQUIRED
----------	------	-------------	----------

PROPERTY	TYPE	DESCRIPTION	REQUIRED
<code>true</code>	string	The color to use when the <code>state</code> value is <code>true</code> . The <code>color</code> property is a JSON string in any one of following formats: <ul style="list-style-type: none"> • HTML-style hex values • RGB ("#ff00", "#ffff00", "rgb(255, 255, 0)") • RGBA ("rgba(255, 255, 0, 1)") • HSL("hsl(100, 50%, 50%)") • HSLA("hsla(100, 50%, 50%, 1)") • Predefined HTML colors names, like yellow, and blue. 	Yes
<code>false</code>	string	The color to use when the <code>state</code> value is <code>false</code> .	Yes

Example of BooleanTypeStyleRule

The following JSON illustrates a `BooleanTypeStyleRule` `state` named `occupied`. The `BooleanRuleObject` defines colors for `true` and `false` values.

```
{
  "keyname": "occupied",
  "type": "boolean",
  "rules": [
    {
      "true": "#FF0000",
      "false": "#00FF00"
    }
  ]
}
```

Consumption model

11/2/2020 • 2 minutes to read • [Edit Online](#)

The Routing service provides a set of parameters for a detailed description of the vehicle-specific Consumption Model. Depending on the value of **vehicleEngineType**, two principal Consumption Models are supported: *Combustion* and *Electric*. It's incorrect to specify parameters that belong to different models in the same request. Also, Consumption Model parameters can't be used with the following **travelMode** values: *bicycle* and *pedestrian*.

Parameter constraints for consumption model

In both Consumption Models, there are some dependencies when specifying parameters. Meaning that, explicitly specifying some parameters may require specifying some other parameters. Here are these dependencies to be aware of:

- All parameters require **constantSpeedConsumption** to be specified by the user. It is an error to specify any other consumption model parameter, if **constantSpeedConsumption** is not specified. The **vehicleWeight** parameter is an exception for this requirement.
- **accelerationEfficiency** and **decelerationEfficiency** must always be specified as a pair (that is, both or none).
- If **accelerationEfficiency** and **decelerationEfficiency** are specified, product of their values must not be greater than 1 (to prevent perpetual motion).
- **uphillEfficiency** and **downhillEfficiency** must always be specified as a pair (that's, both or none).
- If **uphillEfficiency** and **downhillEfficiency** are specified, product of their values must not be greater than 1 (to prevent perpetual motion).
- If the *Efficiency parameters are specified by the user, then **vehicleWeight** must also be specified. When **vehicleEngineType** is *combustion*, **fuelEnergyDensityInMJoulesPerLiter** must be specified as well.
- **maxChargeInkWh** and **currentChargeInkWh** must always be specified as a pair (that is, both or none).

NOTE

If only **constantSpeedConsumption** is specified, no other consumption aspects like slopes and vehicle acceleration are taken into account for consumption computations.

Combustion consumption model

The Combustion Consumption Model is used when **vehicleEngineType** is set to *combustion*. The list of parameters that belong to this model are below. Refer to the Parameters section for detailed description.

- **constantSpeedConsumptionInLitersPerHundredkm**
- **vehicleWeight**
- **currentFuelInLiters**
- **auxiliaryPowerInLitersPerHour**
- **fuelEnergyDensityInMJoulesPerLiter**
- **accelerationEfficiency**
- **decelerationEfficiency**
- **uphillEfficiency**
- **downhillEfficiency**

Electric consumption model

The Electric Consumption Model is used when `vehicleEngineType` is set to `electric`. The list of parameters that belong to this model are below. Refer to the Parameters section for detailed description.

- `constantSpeedConsumptionInkWhPerHundredkm`
- `vehicleWeight`
- `currentChargeInkWh`
- `maxChargeInkWh`
- `auxiliaryPowerInkW`
- `accelerationEfficiency`
- `decelerationEfficiency`
- `uphillEfficiency`
- `downhillEfficiency`

Sensible values of consumption parameters

A particular set of consumption parameters can be rejected, even though the set might fulfill all the explicit requirements. It happens when the value of a specific parameter, or a combination of values of several parameters, is considered to lead to unreasonable magnitudes of consumption values. If that happens, it most likely indicates an input error, as proper care is taken to accommodate all sensible values of consumption parameters. In case a particular set of consumption parameters is rejected, the accompanying error message will contain a textual explanation of the reason(s). The detailed descriptions of the parameters have examples of sensible values for both models.

Extended GeoJSON geometries

11/2/2020 • 2 minutes to read • [Edit Online](#)

Azure Maps provides a list of powerful APIs to search inside and along geographical features. These APIs adhere to the standard [GeoJSON spec](#) of representing geographical features.

The [GeoJSON spec](#) supports only the following geometries:

- GeometryCollection
- LineString
- MultiLineString
- MultiPoint
- MultiPolygon
- Point
- Polygon

Some Azure Maps APIs accept geometries that aren't part of the [GeoJSON spec](#). For instance, the [Search Inside Geometry](#) API accepts Circle and Polygons.

This article provides a detailed explanation on how Azure Maps extends the [GeoJSON spec](#) to represent certain geometries.

Circle

The `circle` geometry is not supported by the [GeoJSON spec](#). We use a `GeoJSON Point Feature` object to represent a circle.

A `circle` geometry represented using the `GeoJSON Feature` object **must** contain the following coordinates and properties:

- Center

The circle's center is represented using a `GeoJSON Point` object.

- Radius

The circle's `radius` is represented using `GeoJSON Feature`'s properties. The radius value is in *meters* and must be of the type `double`.

- SubType

The circle geometry must also contain the `subType` property. This property must be a part of the `GeoJSON Feature`'s properties and its value should be *Circle*.

Example

Here's how you'll represent a circle using a `GeoJSON Feature` object. Let's center the circle at latitude: 47.639754 and longitude: -122.126986, and assign it a radius equal to 100 meters:

```
{  
    "type": "Feature",  
    "geometry": {  
        "type": "Point",  
        "coordinates": [-122.126986, 47.639754]  
    },  
    "properties": {  
        "subType": "Circle",  
        "radius": 100  
    }  
}
```

Rectangle

The `Rectangle` geometry is not supported by the [GeoJSON spec](#). We use a `GeoJSON Polygon Feature` object to represent a rectangle. The rectangle extension is primarily used by the Web SDK's drawing tools module.

A `Rectangle` geometry represented using the `GeoJSON Polygon Feature` object **must** contain the following coordinates and properties:

- **Corners**

The rectangle's corners are represented using the coordinates of a `GeoJSON Polygon` object. There should be five coordinates, one for each corner. And, a fifth coordinate that is the same as the first coordinate, to close the polygon ring. It will be assumed that these coordinates align, and that the developer may rotate them as wanted.

- **SubType**

The rectangle geometry must also contain the `subType` property. This property must be a part of the `GeoJSON Feature`'s properties, and its value should be `Rectangle`.

Example

```
{  
    "type": "Feature",  
    "geometry": {  
        "type": "Polygon",  
        "coordinates": [[[5,25],[14,25],[14,29],[5,29],[5,25]]]  
    },  
    "properties": {  
        "subType": "Rectangle"  
    }  
}
```

Next steps

Learn more about GeoJSON data in Azure Maps:

[Geofence GeoJSON format](#)

Review the glossary of common technical terms associated with Azure Maps and location intelligence applications:

[Azure Maps glossary](#)

Geofencing GeoJSON data

11/2/2020 • 2 minutes to read • [Edit Online](#)

The Azure Maps [GET Geofence](#) and [POST Geofence](#) APIs allow you to retrieve proximity of a coordinate relative to a provided geofence or set of fences. This article details how to prepare the geofence data that can be used in the Azure Maps GET and POST API.

The data for geofence or set of geofences is represented by `Feature` Object and `FeatureCollection` Object in `GeoJSON` format, which is defined in [rfc7946](#). In Addition to it:

- The GeoJSON Object type can be a `Feature` Object or a `FeatureCollection` Object.
- The Geometry Object type can be a `Point`, `MultiPoint`, `LineString`, `MultiLineString`, `Polygon`, `MultiPolygon`, and `GeometryCollection`.
- All feature properties should contain a `geometryId`, which is used for identifying the geofence.
- Feature with `Point`, `MultiPoint`, `LineString`, `MultiLineString` must contain `radius` in properties. `radius` value is measured in meters, the `radius` value ranges from 1 to 10000.
- Feature with `polygon` and `multipolygon` geometry type does not have a radius property.
- `validityTime` is an optional property that lets the user set expired time and validity time period for the geofence data. If not specified, the data never expires and is always valid.
- The `expiredTime` is the expiration date and time of geofencing data. If the value of `userTime` in the request is later than this value, the corresponding geofence data is considered as expired data and is not queried. Upon which, the `geometryId` of this geofence data will be included in `expiredGeofenceGeometryId` array within the geofence response.
- The `validityPeriod` is a list of validity time period of the geofence. If the value of `userTime` in the request falls outside of the validity period, the corresponding geofence data is considered as invalid and will not be queried. The `geometryId` of this geofence data is included in `invalidPeriodGeofenceGeometryId` array within geofence response. The following table shows the properties of validityPeriod element.

NAME	TYPE	REQUIRED	DESCRIPTION
startTime	Datetime	true	The start date time of the validity time period.
endTime	Datetime	true	The end date time of the validity time period.
recurrenceType	string	false	The recurrence type of the period. The value can be <code>Daily</code> , <code>Weekly</code> , <code>Monthly</code> , or <code>Yearly</code> . Default value is <code>Daily</code> .
businessDayOnly	Boolean	false	Indicate whether the data is only valid during business days. Default value is <code>false</code> .

- All coordinate values are represented as [longitude, latitude] defined in `WGS84`.
- For each Feature, which contains `MultiPoint`, `MultiLineString`, `MultiPolygon`, Or `GeometryCollection`, the

properties are applied to all the elements. for example: All the points in `MultiPoint` will use same radius to form a multiple circle geofence.

- In point-circle scenario, a circle geometry can be represented using a `Point` geometry object with properties elaborated in [Extending GeoJSON geometries](#).

Following is a sample request body for a geofence represented as a circle geofence geometry in `GeoJSON` using a center point and a radius. The valid period of the geofence data starts from 2018-10-22, 9AM to 5PM, repeated every day except for the weekend. `expiredTime` indicates this geofence data will be considered expired, if `userTime` in the request is later than `2019-01-01`.

```
{  
    "type": "Feature",  
    "geometry": {  
        "type": "Point",  
        "coordinates": [-122.126986, 47.639754]  
    },  
    "properties": {  
        "geometryId" : "1",  
        "subType": "Circle",  
        "radius": 500,  
        "validityTime":  
        {  
            "expiredTime": "2019-01-01T00:00:00",  
            "validityPeriod": [  
                {  
                    "startTime": "2018-10-22T09:00:00",  
                    "endTime": "2018-10-22T17:00:00",  
                    "recurrenceType": "Daily",  
                    "recurrenceFrequency": 1,  
                    "businessDayOnly": true  
                }  
            ]  
        }  
    }  
}
```

Azure Maps supported categories

11/2/2020 • 7 minutes to read • [Edit Online](#)

When doing a [category search](#) for points of interest, there are over a hundred supported categories. Below is a list of the category codes for supported category names. Category codes are generated for top-level categories. All sub categories share same category code. This category list is subject to change with new data releases.

CATEGORY CODE	CATEGORIES MATCHING CODE
ACCESS_GATEWAY	airline access, security gate, station access, access gateway
ADMINISTRATIVE_DIVISION	province, fourth-order administrative division, first-order administrative division, historical third-order administrative division, seat of a fourth-order administrative division, seat of a second-order administrative division, dependent political entity, populated place, seat of a third-order administrative division, populated places, second-order administrative division, seat of a first-order administrative division, administrative division, populated locality, historical region, historical site, historical populated place, Israeli settlement, historical fourth-order administrative division, fifth-order administrative division, historical first-order administrative division, third-order administrative division, historical political entity, historical administrative division, seat of government of a political entity, historical second-order administrative division, capital/major city of a political entity
ADVENTURE_SPORTS_VENUE	adventure sports venue
AGRICULTURE	horticulture, primary producer, agriculture, farm, farm village, farmstead, homestead, grazing area, common, aquaculture facility, farms, fishing area, dairy, field(s)
AIRPORT	private authority, military authority, heliport, closed, medium airport, large airport, small airport, airfield, seaplane base, public authority, balloon port, airport
AMUSEMENT_PARK	amusement arcade, amusement place, amusement park
AUTOMOTIVE DEALER	atv/snowmobile, boat, bus, motorcycle, truck, van, recreational vehicles, car, automotive dealer
BANK	bank, banks, bank(s)
BEACH	beach, beaches
BUILDING_POINT	building (point)
BUSINESS_PARK	business park, industrial area

CATEGORY CODE	CATEGORIES MATCHING CODE
CAFE_PUB	internet café, tea house, cafe, internet cafe, café, coffee shop, microbrewery/beer garden, pub, café/pub, cafe/pub
CAMPING_GROUND	recreational, caravan site, camping ground
CAR_WASH	car wash
CASH_DISPENSER	automatic teller machine, cash dispenser
CASINO	casino
CINEMA	drive-in cinema, cinema
CITY_CENTER	neighborhood, administrative area, city center, center
CLUB_ASSOCIATION	beach club, hockey club, club association
COLLEGE_UNIVERSITY	junior college/community college, college/university, college, university prep school, university
COMMERCIAL_BUILDING	office building, park headquarters, commercial building
COMMUNITY_CENTER	community center
COMPANY	electronics, manufacturing, computer data services, public health technologies, diversified financials, animal shelter, airline, equipment rental, service, mail/package/freight delivery, bus lines, home appliance repair, cleaning services, oem, tax services, oil natural gas, legal services, construction, telecommunications, transport, automobile manufacturing, chemicals, funeral service mortuaries, bridge tunnel operations, automobile, mechanical engineering, services, investment advisors, advertising/marketing, moving storage, savings institution, insurance, computer software, pharmaceuticals, catering, wedding services, agricultural technology, real estate, taxi, limousine shuttle service, bus charter rentals, mining, publishing technologies, cable telephone, import/export distribution, company, asylum, coal mine(s), estate(s), brewery, gold mine(s)
COURTHOUSE	courthouse
CULTURAL_CENTER	cultural center
DENTIST	dentist
DEPARTMENT_STORE	department store
DOCTOR	general practitioner, specialist, doctor
ELECTRIC_VEHICLE_STATION	electric vehicle station
EMBASSY	embassy

CATEGORY CODE	CATEGORIES MATCHING CODE
EMERGENCY_MEDICAL_SERVICE	emergency medical service
ENTERTAINMENT	entertainment
EXCHANGE	gold exchange, currency exchange, stock exchange, exchange
EXHIBITION_CONVENTION_CENTER	exhibition convention center
FERRY_TERMINAL	ferry, ferry terminal
FIRE_STATION_BRIGADE	fire station/brigade
FRONTIER_CROSSING	frontier crossing
FUEL_FACILITIES	fuel facilities
GEOGRAPHIC_FEATURE	bay, cove, pan, locale, ridge, mineral/hot springs, well, reservoir, marsh/swamp/vlei, quarry, river crossing, valley, mountain peak, reef, dune, lagoon, plain/flat, rapids, cape, plateau, oasis, harbor, cave, rocks, geographic feature, promontory(-ies), islands, headland, pier, crater lake, cliff(s), hill, desert, portage, glacier(s), gully, geyser, coral reef(s), gap, gulf, jetty, ghat, hole, crater lakes, gas field, islet, crater(s), cove(s), grassland, gravel area, fracture zone, heath, gorge(s), island, headwaters, hanging valley, hills, hot spring(s), furrow, anabanch
GOLF.Course	golf course
GOVERNMENT_OFFICE	order 5 area, order 8 area, order 9 area, order 2 area, order 7 area, order 3 area, supra national, order 4 area, order 6 area, government office, diplomatic facility, united states government establishment, local government office, customs house, customs post
HEALTH_CARE_SERVICE	blood bank, personal service, personal care facility, ambulance unit, health care service, leprosarium, sanatorium, hospital, medical center, clinic
HELIPAD_HELIICOPTER_LANDING	helipad/helicopter landing
HOLIDAY_RENTAL	bungalow, cottage, chalet, villa, apartment, holiday rental
HOSPITAL_POLYCLINIC	special, hospital of Chinese medicine, hospital for women children, general, hospital/polyclinic
HOTEL_MOTEL	cabins lodges, bed breakfast guest houses, hotel, rest camps, motel, resort, hostel, hotel/motel, resthouse, hammock(s), guest house
ICE_SKATING_RINK	ice skating rink

CATEGORY CODE	CATEGORIES MATCHING CODE
IMPORTANT_TOURIST_ATTRACTION	building, observatory, arch, tunnel, statue, tower, bridge, planetarium, mausoleum/grave, monument, water hole, natural attraction, important tourist attraction, promenade, pyramids, pagoda, castle, palace, hermitage, pyramid, fort, gate, country house, dam, lighthouse, grave
INDUSTRIAL_BUILDING	foundry, fuel depot, industrial building, factory
LEISURE_CENTER	bowling, snooker, pool billiard, flying club, dance studio school, sauna, solarium massage, leisure center, spa
LIBRARY	library
MANUFACTURING_FACILITY	manufacturing facility
MARINA	yacht basin, marina
MARKET	supermarkets hypermarkets, farmers, public, informal, market
MEDIA_FACILITY	media facility
MILITARY_INSTALLATION	military base, coast guard station, military installation, naval base
MOTORING_ORGANIZATION_OFFICE	motoring organization office
MOUNTAIN_PASS	mountain pass
MUSEUM	museum
NATIVE_RESERVATION	native reservation, reservation
NIGHTLIFE	bar, karaoke club, jazz club, private club, wine bar, comedy club, cocktail bar, discotheque, nightlife
NON_GOVERNMENTAL_ORGANIZATION	non-governmental organization
OPEN_PARKING_AREA	open parking area, parking lot
OTHER	locality, free trade zone, traffic circle, unknown
PARKING_GARAGE	parking garage
PARK_RECREATION_AREA	historic site, lakeshore, seashore, river scenic area, fishing hunting area, battlefield, winter sport, boat launching ramp, preserve, forest area, recreation area, ski resort, cemetery, historical park, parkway, memorial, fairground, picnic area, wilderness area, park recreation area, forest(s), fossilized forest, garden(s), wildlife reserve, nature reserve, forest station, hunting reserve, forest reserve, park
PETROL_STATION	petrol station

CATEGORY CODE	CATEGORIES MATCHING CODE
PHARMACY	pharmacy, dispensary
PLACE_OF_WORSHIP	ashram, synagogue, mosque, gurudwara, church, temple, place of worship, mission, retreat, temple(s), religious site, religious center, monastery, convent
POLICE_STATION	order 1 area, police station, police post
PORT_WAREHOUSE_FACILITY	harbor(s), docking basin, port, port/warehouse facility, dockyard, dock(s)
POST_OFFICE	local, post office
PRIMARY_RESOURCE.Utility	primary resource/utility, power station, gas-oil separator plant
PRISON_CORRECTIONAL.FACILITY	prison, prison/correctional facility
PUBLIC_AMENITY	pedestrian subway, toilet, road rescue, passenger transport ticket office, public call box, public amenity, communication center
PUBLIC_TRANSPORT_STOP	coach stop, bus stop, taxi stand, tram stop, public transport stop, metro station, railroad station, bus station, railroad stop
RAILWAY_STATION	national, railway siding, metro, (sub) urban, railway station
RENT_A_CAR.FACILITY	rent-a-car facility
RENT_A_CAR.PARKING	rent-a-car-parking
REPAIR.FACILITY	bodyshops, tyre (tire) services, repair shops, car glass replacement shops, general car repair servicing, sale installation of car accessories, motorcycle repair, truck repair service, repair facility
RESEARCH.FACILITY	research facility
RESIDENTIAL_ACCOMMODATION	retirement community, townhouse complex, flats/apartment complex, condominium complex, residential estate, residential accommodation

CATEGORY CODE	CATEGORIES MATCHING CODE
RESTAURANT	German, creole-Cajun, Dutch, banquet rooms, bistro, Israeli, Slovak, Jamaican, vegetarian, seafood, Vietnamese, Maltese, Sichuan, welsh, Chinese, Japanese, Algerian, Californian, fusion, Shandong, salad bar, Savoy an, Spanish, Ethiopian, Taiwanese, doughnuts, Iranian, Canadian, American, Norwegian, French, Hunan, Polynesian, afghan, roadside, Asian, swiss, erotic, crêperie, Surinamese, Egyptian, Hungarian, Nepalese, barbecue, hot pot, hamburgers, Mediterranean, Latin American, tapas, British, Mexican, Guangdong, Asian (other), buffet, sushi, Mongolian, international, mussels, Thai, Venezuelan, Rumanian, chicken, soup, kosher, steak house, yogurt/juice bar, Italian, Korean, Cypriot, Bosnian, Bolivian, Dominican, Belgian, Tunisian, Scottish, English, Pakistani, Czech, Hawaiian, Maghrib, Tibetan, Arabian, middle eastern, Chilean, Shanghai, polish, Filipino, Sudanese, Armenian, Burmese, Brazilian, Scandinavian, Bulgarian, soul food, Colombian, Jewish, pizza, Sicilian, organic, Greek, Basque, Uruguayan, cafeterias, Finnish, African, Corsican, Syrian, Caribbean, Dongbei, Russian, grill, take away, fast food, Australian, Irish, pub food, fondue, Lebanese, Indonesian, Danish, Provençal, teppanyaki, Indian, Mauritian, western continental, Peruvian, Cambodian, snacks, Swedish, macrobiotic, ice cream parlor, Slavic, Turkish, Argentinean, Austrian, exotic, Portuguese, Luxembourgian, Moroccan, sandwich, Cuban, restaurant
RESTAURANT_AREA	restaurant area
REST_AREA	rest area, halting place
SCENIC_PANORAMIC_VIEW	scenic/panoramic view, observation point
SCHOOL	culinary school, primary school, art school, senior high school, driving school, language school, sport school, preschool, high school, middle school, vocational training, special school, child care facility, school, technical school, military school, agricultural school

CATEGORY CODE	CATEGORIES MATCHING CODE
SHOP	factory outlet, security products, Christmas/holiday store, opticians, house garden: lighting, lottery shop, musical instruments, nail salon, house garden: painting/decorating, hobby/free time, newsagents/tobacconists, clothing accessories: specialty, dry cleaners, bags/leatherwear, pet supplies, clothing accessories: children, construction material/equipment, jewelry, clocks/watches, clothing accessories: footwear/shoe repairs, house garden: curtains/textiles, electrical, office it: consumer electronics, electrical, office it: camera's/photography, cd's/DVD/videos, laundry, clothing accessories: men, florists, pawn shop, book shops, marine/electronic equipment, food/drinks: food markets, house garden: carpet/floor coverings, photocopy, boating equipment/accessories, mobile phone shop, toys/games, specialty foods, clothing accessories: general, food/drinks: bakers, tailor shop, gifts, cards, novelties/souvenirs, animal services, sports equipment/clothing, stamp shop, electrical appliance/electrical, office it: office equipment, photo lab/development, wholesale clubs, house garden: furniture fittings, local specialties, food/drinks: butchers, variety store, food/drinks: food shops, food/drinks: wine/spirits, drug store, furniture/home furnishings, electrical, office it: computer supplies, cd/video rental, medical supplies/equipment, agricultural supplies, beauty salon, house garden: garden centers/services, food/drinks: fishmongers, beauty supplies, clothing accessories: women, travel agents, retail outlet, recycling shop, house garden: glass/windows, hardware, real estate agents, glassware/ceramic, delicatessen, house garden: kitchens/bathrooms, betting station, hairdressers/barbers, food/drinks: grocers, food/drinks: green/grocers, convenience stores, drive-through/bottle shop, house garden: do-it-yourself centers, antique/art, shop, store
SHOPPING_CENTER	mall, shopping center
SPORTS_CENTER	thematic sport, squash court, fitness club/center, sports center
STADIUM	netball, football, baseball, race track, multi-purpose/motor sport, cricket ground, rugby ground, ice hockey, athletic, horse racing, basketball, soccer, stadium, athletic field, racetrack
SWIMMING_POOL	swimming pool
TENNIS_COURT	tennis court
THEATER	amphitheater, concert hall, dinner theater, music center, opera, cabaret, theater, opera house
TOURIST_INFORMATION_OFFICE	tourist information office
TRAFFIC_LIGHT	traffic light
TRAFFIC_SERVICE_CENTER	traffic control department, traffic service center
TRAFFIC_SIGN	traffic sign

CATEGORY CODE	CATEGORIES MATCHING CODE
TRAIL_SYSTEM	adventure vehicle, rock climbing, horse riding, mountain bike, hiking, trail system
TRANSPORT_AUTHORITY VEHICLE_REGISTRATION	transport authority/vehicle registration
TRUCK_STOP	truck stop
VETERINARIAN	veterinary facility, veterinarian
WATER_SPORT	water sport
WEIGH_STATION	weigh scales, weigh station
WELFARE_ORGANIZATION	welfare organization
WINERY	winery
ZOOS_ARBORETA_BOTANICAL_GARDEN	wildlife park, aquatic zoo marine park, arboreta botanical gardens, zoo, zoos, arboreta botanical garden

Facility Ontology

6/8/2021 • 20 minutes to read • [Edit Online](#)

Facility ontology defines how Azure Maps Creator internally stores facility data in a Creator dataset. In addition to defining internal facility data structure, facility ontology is also exposed externally through the WFS API. When WFS API is used to query facility data in a dataset, the response format is defined by the ontology supplied to that dataset.

At a high level, facility ontology divides the dataset into feature classes. All feature classes share a common set of properties, such as `ID` and `Geometry`. In addition to the common property set, each feature class defines a set of properties. Each property is defined by its data type and constraints. Some feature classes have properties that are dependent on other feature classes. Dependant properties evaluate to the `ID` of another feature class.

Changes and Revisions

The Facility 1.0 contains revisions for the Facility feature class definitions for [Azure Maps Services](#).

The Facility 2.0 contains revisions for the Facility feature class definitions for [Azure Maps Services](#).

Major Changes

Fixed the following constraint validation checks:

- Constraint validation check for exclusivity of `isObstruction = true` or the presence of `obstructionArea` for `lineElement` and `areaElement` feature classes.
- Constraint validation check for exclusivity of `isRoutable = true` or the presence of `routeThroughBehavior` for the `category` feature class.
- Added a structure feature class to hold walls, columns, and so on.
- Cleaned up the attributes designed to enrich routing scenarios. The current routing engine doesn't support them.

unit

The `unit` feature class defines a physical and non-overlapping area that can be occupied and traversed by a navigating agent. A `unit` can be a hallway, a room, a courtyard, and so on.

Geometry Type: Polygon

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>categoryId</code>	<code>category.Id</code>	true	The ID of a <code>category</code> feature.
<code>isOpenArea</code>	boolean (Default value is <code>null</code> .)	false	Represents whether the unit is an open area. If set to <code>true</code> , <code>structures</code> don't surround the unit boundary, and a navigating agent can enter the <code>unit</code> without the need of an <code>opening</code> . By default, units are surrounded by physical barriers and are open only where an opening feature is placed on the boundary of the unit. If walls are needed in an open area unit, they can be represented as a <code>lineElement</code> or <code>areaElement</code> with an <code>isObstruction</code> property equal to <code>true</code> .
<code>navigableBy</code>	enum ["pedestrian", "wheelchair", "machine", "bicycle", "automobile", "hiredAuto", "bus", "railcar", "emergency", "ferry", "boat"]	false	Indicates the types of navigating agents that can traverse the unit. If unspecified, the unit is assumed to be traversable by any navigating agent.
<code>isRoutable</code>	boolean (Default value is <code>null</code> .)	false	Determines if the unit is part of the routing graph. If set to <code>true</code> , the unit can be used as source/destination or intermediate node in the routing experience.
<code>routeThroughBehavior</code>	enum ["disallowed", "allowed", "preferred"]	false	Determines if navigating through the unit is allowed. If unspecified, it inherits its value from the category feature referred to in the <code>categoryId</code> property. If specified, it overrides the value given in its category feature."
<code>nonPublic</code>	boolean	false	If <code>true</code> , the unit is navigable only by privileged users. Default value is <code>false</code> .
<code>levelId</code>	<code>level.Id</code>	true	The ID of a level feature.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>occupants</code>	array of directoryInfo.id	false	The IDs of directoryInfo features. Used to represent one or many occupants in the feature.
<code>addressId</code>	directoryInfo.id	true	The ID of a directoryInfo feature. Used to represent the address of the feature.
<code>addressRoomNumber</code>	directoryInfo.id	true	Room/Unit/Apartment/Suite number of the unit.
<code>name</code>	string	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	string	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.
<code>anchorPoint</code>	Point	false	GeoJSON Point geometry that represents the feature as a point. Can be used to position the label of the feature.
PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	category.id	true	The ID of a category feature.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>isOpenArea</code>	boolean (Default value is <code>null</code> .)	false	Represents whether the unit is an open area. If set to <code>true</code> , structures don't surround the unit boundary, and a navigating agent can enter the <code>unit</code> without the need of an <code>opening</code> . By default, units are surrounded by physical barriers and are open only where an opening feature is placed on the boundary of the unit. If walls are needed in an open area unit, they can be represented as a <code>lineElement</code> or <code>areaElement</code> with an <code>isObstruction</code> property equal to <code>true</code> .
<code>isRoutable</code>	boolean (Default value is <code>null</code> .)	false	Determines if the unit is part of the routing graph. If set to <code>true</code> , the unit can be used as source/destination or intermediate node in the routing experience.
<code>levelId</code>	<code>level.Id</code>	true	The ID of a level feature.
<code>occupants</code>	array of <code>directoryInfo.Id</code>	false	The IDs of <code>directoryInfo</code> features. Used to represent one or many occupants in the feature.
<code>addressId</code>	<code>directoryInfo.Id</code>	true	The ID of a <code>directoryInfo</code> feature. Used to represent the address of the feature.
<code>addressRoomNumber</code>	<code>directoryInfo.Id</code>	true	Room/Unit/Apartment/Suite number of the unit.
<code>name</code>	string	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	string	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>anchorPoint</code>	Point	false	GeoJSON Point geometry that represents the feature as a point. Can be used to position the label of the feature.

structure

The `structure` feature class defines a physical and non-overlapping area that cannot be navigated through. Can be a wall, column, and so on.

Geometry Type: Polygon

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	<code>category.id</code>	true	The ID of a <code>category</code> feature.
<code>levelId</code>	<code>level.id</code>	true	The ID of a <code>level</code> feature.
<code>name</code>	string	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	string	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.
<code>anchorPoint</code>	Point	false	GeoJSON Point geometry that represents the feature as a point. Can be used to position the label of the feature.

zone

The `zone` feature class defines a virtual area, like a WiFi zone or emergency assembly area. Zones can be used as destinations but are not meant for through traffic.

Geometry Type: Polygon

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	<code>category.id</code>	true	The ID of a <code>category</code> feature.
<code>setId</code>	string	true	Required for zone features that represent multi-level zones. The <code>setId</code> is the unique ID for a zone that spans multiple levels. The <code>setId</code> enables a zone with varying coverage on different floors to be represented with different geometry on different levels. The <code>setId</code> can be any string and is case-sensitive. It is recommended that the <code>setId</code> is a GUID. Maximum length allowed is 1000.
<code>levelId</code>	<code>level.id</code>	true	The ID of a <code>level</code> feature.
<code>name</code>	string	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	string	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>anchorPoint</code>	<code>Point</code>	false	<code>GeoJSON Point geometry</code> that represents the feature as a point. Can be used to position the label of the feature.

level

The `level` class feature defines an area of a building at a set elevation. For example, the floor of a building, which contains a set of features, such as `units`.

Geometry Type: MultiPolygon

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	<code>string</code>	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	<code>string</code>	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	<code>category.id</code>	true	The ID of a <code>category</code> feature.
<code>ordinal</code>	<code>integer</code>	true	The level number. Used by the <code>verticalPenetration</code> feature to determine the relative order of the floors to help with travel direction. The general practice is to start with 0 for the ground floor. Add +1 for every floor upwards, and -1 for every floor going down. It can be modeled with any numbers, as long as the higher physical floors are represented by higher ordinal values.
<code>abbreviatedName</code>	<code>string</code>	false	A four-character abbreviated level name, like what would be found on an elevator button. Maximum length allowed is 1000.
<code>heightAboveFacilityAnchor</code>	<code>double</code>	false	Vertical distance of the level's floor above <code>facility.anchorHeightAboveSeaLevel</code> , in meters.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>verticalExtent</code>	double	false	Vertical extent of the level, in meters. If not provided, defaults to facility.defaultLevelVerticalExtent .
<code>name</code>	string	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	string	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.
<code>anchorPoint</code>	Point	false	GeoJSON Point geometry that represents the feature as a point. Can be used to position the label of the feature.

facility

The `facility` feature class defines the area of the site, building footprint, and so on.

Geometry Type: MultiPolygon

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	category.Id	true	The ID of a <code>category</code> feature.
<code>occupants</code>	array of directoryInfo.Id	false	The IDs of <code>directoryInfo</code> features. Used to represent one or many occupants in the feature.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>addressId</code>	<code>directoryInfo.id</code>	true	The ID of a <code>directoryInfo</code> feature. Used to represent the address of the feature.
<code>addressRoomNumber</code>	<code>directoryInfo.id</code>	true	Room/Unit/Apartment/Suite number of the unit.
<code>name</code>	string	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	string	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.
<code>anchorPoint</code>	<code>Point</code>	false	<code>GeoJSON Point geometry</code> that represents the feature as a point. Can be used to position the label of the feature.
<code>anchorHeightAboveSeaLevel</code>	double	false	Height of anchor point above sea level, in meters. Sea level is defined by EGM 2008.
<code>defaultLevelVerticalExtent</code>	double	false	Default value for vertical extent of levels, in meters.

verticalPenetration

The `verticalPenetration` class feature defines an area that, when used in a set, represents a method of navigating vertically between levels. It can be used to model stairs, elevators, and so on. Geometry can overlap units and other vertical penetration features.

Geometry Type: Polygon

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	<code>category.id</code>	true	The ID of a <code>category</code> feature.
<code>setId</code>	string	true	Vertical penetration features must be used in sets to connect multiple levels. Vertical penetration features in the same set are considered to be the same. The <code>setId</code> can be any string, and is case-sensitive. Using a GUID as a <code>setId</code> is recommended. Maximum length allowed is 1000.
<code>levelId</code>	<code>level.id</code>	true	The ID of a level feature.
<code>direction</code>	string enum ["both", "lowToHigh", "highToLow", "closed"]	true	Travel direction allowed on this feature. The ordinal attribute on the <code>level</code> feature is used to determine the low and high order.
<code>navigableBy</code>	enum ["pedestrian", "wheelchair", "machine", "bicycle", "automobile", "hiredAuto", "bus", "railcar", "emergency", "ferry", "boat"]	false	Indicates the types of navigating agents that can traverse the unit. If unspecified, the unit is traversable by any navigating agent.
<code>nonPublic</code>	boolean	false	If <code>true</code> , the unit is navigable only by privileged users. Default value is <code>false</code> .
<code>name</code>	string	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	string	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>anchorPoint</code>	<code>Point</code>	false	GeoJSON Point geometry that represents the feature as a point. Can be used to position the label of the feature.
PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	<code>string</code>	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	<code>string</code>	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	<code>category.id</code>	true	The ID of a category feature.
<code>setId</code>	<code>string</code>	true	Vertical penetration features must be used in sets to connect multiple levels. Vertical penetration features in the same set are connected. The <code>setId</code> can be any string, and is case-sensitive. Using a GUID as a <code>setId</code> is recommended. Maximum length allowed is 1000.
<code>levelId</code>	<code>level.id</code>	true	The ID of a level feature.
<code>direction</code>	<code>string enum ["both", "lowToHigh", "highToLow", "closed"]</code>	true	Travel direction allowed on this feature. The ordinal attribute on the <code>level</code> feature is used to determine the low and high order.
<code>name</code>	<code>string</code>	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	<code>string</code>	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.
<code>anchorPoint</code>	Point	false	GeoJSON Point geometry that represents the feature as a point. Can be used to position the label of the feature.

opening

The `opening` class feature defines a traversable boundary between two units, or a `unit` and `verticalPenetration`.

Geometry Type: LineString

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	<code>category.id</code>	true	The ID of a category feature.
<code>levelId</code>	<code>level.id</code>	true	The ID of a level feature.
<code>isConnectedToVerticalPenetration</code>	boolean	false	Whether or not this feature is connected to a <code>verticalPenetration</code> feature on one of its sides. Default value is <code>false</code> .
<code>navigableBy</code>	enum ["pedestrian", "wheelchair", "machine", "bicycle", "automobile", "hiredAuto", "bus", "railcar", "emergency", "ferry", "boat"]	false	Indicates the types of navigating agents that can traverse the unit. If unspecified, the unit is traversable by any navigating agent.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>accessRightToLeft</code>	enum ["prohibited", "digitalKey", "physicalKey", "keyPad", "guard", "ticket", "fingerprint", "retina", "voice", "face", "palm", "iris", "signature", "handGeometry", "time", "ticketChecker", "other"]	false	Method of access when passing through the opening from right to left. Left and right are determined by the vertices in the feature geometry, standing at the first vertex and facing the second vertex. Omitting this property means there are no access restrictions.
<code>accessLeftToRight</code>	enum ["prohibited", "digitalKey", "physicalKey", "keyPad", "guard", "ticket", "fingerprint", "retina", "voice", "face", "palm", "iris", "signature", "handGeometry", "time", "ticketChecker", "other"]	false	Method of access when passing through the opening from left to right. Left and right are determined by the vertices in the feature geometry, standing at the first vertex and facing the second vertex. Omitting this property means there are no access restrictions.
<code>isEmergency</code>	boolean	false	If <code>true</code> , the opening is navigable only during emergencies. Default value is <code>false</code> .
<code>anchorPoint</code>	Point	false	GeoJSON Point geometry that represents the feature as a point. Can be used to position the label of the feature.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	<code>category.id</code>	true	The ID of a category feature.
<code>levelId</code>	<code>level.id</code>	true	The ID of a level feature.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>anchorPoint</code>	Point	false	GeoJSON Point geometry that represents the feature as a point. Can be used to position the label of the feature.

directoryInfo

The `directoryInfo` object class feature defines the name, address, phone number, website, and hours of operation for a unit, facility, or an occupant of a unit or facility.

Geometry Type: None

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>streetAddress</code>	string	false	Street address part of the address. Maximum length allowed is 1000.
<code>unit</code>	string	false	Unit number part of the address. Maximum length allowed is 1000.
<code>locality</code>	string	false	The locality of the address. For example: city, municipality, village). Maximum length allowed is 1000.
<code>adminDivisions</code>	string	false	Administrative division part of the address, from smallest to largest (County, State, Country). For example: ["King", "Washington", "USA"] or ["West Godavari", "Andhra Pradesh", "IND"]. Maximum length allowed is 1000.
<code>postalCode</code>	string	false	Postal code part of the address. Maximum length allowed is 1000.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>name</code>	string	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	string	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.
<code>phoneNumber</code>	string	false	Phone number.
<code>website</code>	string	false	Website URL. Maximum length allowed is 1000.
<code>hoursOfOperation</code>	string	false	Hours of operation as text, following the Open Street Map specification . Maximum length allowed is 1000.

pointElement

The `pointElement` is a class feature that defines a point feature in a unit, such as a first aid kit or a sprinkler head.

Geometry Type: MultiPoint

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	category.id	true	The ID of a category feature.
<code>unitId</code>	string	true	The ID of a unit feature containing this feature. Maximum length allowed is 1000.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>isObstruction</code>	boolean (Default value is <code>null</code> .)	false	If <code>true</code> , this feature represents an obstruction to be avoided while routing through the containing unit feature.
<code>name</code>	string	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	string	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.

lineElement

The `lineElement` is a class feature that defines a line feature in a unit, such as a dividing wall or window.

Geometry Type: LinearMultiString

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	<code>category.id</code>	true	The ID of a <code>category</code> feature.
<code>unitId</code>	string	true	The ID of a <code>unit</code> feature containing this feature. Maximum length allowed is 1000.
<code>isObstruction</code>	boolean (Default value is <code>null</code> .)	false	If <code>true</code> , this feature represents an obstruction to be avoided while routing through the containing unit feature.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>name</code>	string	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	string	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.
<code>anchorPoint</code>	Point	false	GeoJSON Point geometry that represents the feature as a point. Can be used to position the label of the feature.
<code>obstructionArea</code>	Polygon	false	A simplified geometry (when the line geometry is complicated) of the feature that is to be avoided during routing. Requires <code>isObstruction</code> set to true.

areaElement

The `areaElement` is a class feature that defines a polygon feature in a unit, such as an area open to below, an obstruction like an island in a unit.

Geometry Type: MultiPolygon

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the feature with another feature in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>categoryId</code>	category.id	true	The ID of a category feature.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>unitId</code>	string	true	The ID of a <code>unit</code> feature containing this feature. Maximum length allowed is 1000.
<code>isObstruction</code>	boolean	false	If <code>true</code> , this feature represents an obstruction to be avoided while routing through the containing unit feature.
<code>obstructionArea</code>	geometry: ["Polygon", "MultiPolygon"]	false	A simplified geometry (when the line geometry is complicated) of the feature that is to be avoided during routing. Requires <code>isObstruction</code> set to true.
<code>name</code>	string	false	Name of the feature in local language. Maximum length allowed is 1000.
<code>nameSubtitle</code>	string	false	Subtitle that shows up under the <code>name</code> of the feature. Can be used to display the name in a different language, and so on. Maximum length allowed is 1000.
<code>nameAlt</code>	string	false	Alternate name used for the feature. Maximum length allowed is 1000.
<code>anchorPoint</code>	Point	false	GeoJSON Point geometry that represents the feature as a point. Can be used to position the label of the feature.

category

The `category` class feature defines category names. For example: "room.conference".

Geometry Type: None

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The category's original ID derived from client data. Maximum length allowed is 1000.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>externalId</code>	string	true	An ID used by the client to associate the category with another category in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>name</code>	string	true	Name of the category. Suggested to use "." to represent hierarchy of categories. For example: "room.conference", "room.privateoffice". Maximum length allowed is 1000.
<code>routeThroughBehavior</code>	boolean	false	Determines whether a feature can be used for through traffic.
<code>isRoutable</code>	boolean (Default value is <code>null</code> .)	false	Determines if a feature should be part of the routing graph. If set to <code>true</code> , the unit can be used as source/destination or intermediate node in the routing experience.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
<code>originalId</code>	string	true	The category's original ID derived from client data. Maximum length allowed is 1000.
<code>externalId</code>	string	true	An ID used by the client to associate the category with another category in a different dataset, such as in an internal database. Maximum length allowed is 1000.
<code>name</code>	string	true	Name of the category. Suggested to use "." to represent hierarchy of categories. For example: "room.conference", "room.privateoffice". Maximum length allowed is 1000.

What are the Azure IoT support and help options?

11/2/2020 • 2 minutes to read • [Edit Online](#)

Here are suggestions for where you can get help when developing your Azure IoT solutions.

Create an Azure support request



Explore the range of [Azure support options and choose the plan](#) that best fits, whether you're a developer just starting your cloud journey or a large organization deploying business-critical, strategic applications. Azure customers can create and manage support requests in the Azure portal.

- [Azure portal](#)
- [Azure portal for the United States government](#)

Post a question on Microsoft Q&A

For quick and reliable answers on your technical product questions from Microsoft Engineers, Azure Most Valuable Professionals (MVPs), or our expert community, engage with us on [Microsoft Q&A](#), Azure's preferred destination for community support.

If you can't find an answer to your problem using search, submit a new question to Microsoft Q&A. Use one of the following tags when you ask your question:

- [Azure IoT](#)
- [Azure IoT Central](#)
- [Azure IoT Edge](#)
- [Azure IoT Hub](#)
- [Azure IoT Hub Device Provisioning Service \(DPS\)](#)
- [Azure IoT SDKs](#)
- [Azure Digital Twins](#)
- [Azure RTOS](#)
- [Azure Sphere](#)
- [Azure Time Series Insights](#)
- [Azure Maps](#)

Post a question on Stack Overflow



For answers on your developer questions from the largest community developer ecosystem, ask your question on Stack Overflow.

If you do submit a new question to Stack Overflow, please use one or more of the following tags when you create the question:

- [Azure IoT Central](#)
- [Azure IoT Edge](#)
- [Azure IoT Hub](#)
- [Azure IoT SDKs](#)

- [Azure Digital Twins](#)
- [Azure RTOS](#)
- [Azure Sphere](#)
- [Azure Time Series Insights](#)
- [Azure Maps](#)

Submit feedback on Azure Feedback

To request new features, post them on Azure Feedback. Share your ideas for making Azure IoT services work better for the applications you develop:

SERVICE	AZURE FEEDBACK URL
Azure IoT (Hub, DPS, SDKs)	https://feedback.azure.com/forums/321918-azure-iot
Azure IoT Central	https://feedback.azure.com/forums/911455-azure-iot-central
Azure IoT Device Catalog	https://feedback.azure.com/forums/916948-azure-iot-device-catalog
Azure IoT Edge	https://feedback.azure.com/forums/907045-azure-iot-edge
Azure IoT Solution Accelerators	https://feedback.azure.com/forums/916438-azure-iot-solution-accelerators
Azure Maps	https://feedback.azure.com/forums/909172-azure-maps
Azure Time Series Insights	https://feedback.azure.com/forums/906859-azure-time-series-insights
Azure Digital Twins	https://feedback.azure.com/forums/916621-azure-digital-twins
Azure Sphere	https://feedback.azure.com/forums/915433-azure-sphere

Stay informed of updates and new releases

Learn about important product updates, roadmap, and announcements in [Azure Updates](#).

News and information about Azure IoT is shared at the [Azure blog](#) and on the [Internet of Things Show on Channel 9](#).

Also, share your experiences, engage and learn from experts in the [Internet of Things Tech Community](#).

Next steps

[What is Azure IoT?](#)

Azure IoT Hub SDKs

6/8/2021 • 2 minutes to read • [Edit Online](#)

There are two categories of software development kits (SDKs) for working with IoT Hub:

- **IoT Hub Service SDKs** enable you to build backend applications to manage your IoT hub, and optionally send messages, schedule jobs, invoke direct methods, or send desired property updates to your IoT devices or modules.
- **IoT Hub Device SDKs** enable you to build apps that run on your IoT devices using device client or module client. These apps send telemetry to your IoT hub, and optionally receive messages, job, method, or twin updates from your IoT hub. You can use these SDKs to build device apps that use [Azure IoT Plug and Play](#) conventions and models to advertise their capabilities to IoT Plug and Play-enabled applications. You can also use module client to author [modules](#) for [Azure IoT Edge runtime](#).

In addition, we also provide a set of SDKs for working with the [Device Provisioning Service](#).

- **Provisioning Device SDKs** enable you to build apps that run on your IoT devices to communicate with the Device Provisioning Service.
- **Provisioning Service SDKs** enable you to build backend applications to manage your enrollments in the Device Provisioning Service.

Learn about the [benefits of developing using Azure IoT SDKs](#).

Azure IoT Hub Service SDKs

The Azure IoT service SDKs contain code to facilitate building applications that interact directly with IoT Hub to manage devices and security.

PLATFORM	PACKAGE	CODE REPOSITORY	SAMPLES	REFERENCE
.NET	NuGet	GitHub	Samples	Reference
Java	Maven	GitHub	Samples	Reference
Node	npm	GitHub	Samples	Reference
Python	pip	GitHub	Samples	Reference
Node.js	npm	GitHub	Samples	Reference

Azure IoT Hub service SDK for iOS:

- Install from [CocoaPod](#)
- [Samples](#)

Microsoft Azure Provisioning SDKs

The [Microsoft Azure Provisioning SDKs](#) enable you to provision devices to your IoT Hub using the [Device Provisioning Service](#).

Platform	Package	Source Code	Reference
.NET	Device SDK, Service SDK	GitHub	Reference
C	apt-get, MBED, Arduino IDE or iOS	GitHub	Reference
Java	Maven	GitHub	Reference
Node.js	Device SDK, Service SDK	GitHub	Reference
Python	Device SDK, Service SDK	GitHub	Device Reference, Service Reference

Azure IoT Hub Device SDKs

The Microsoft Azure IoT device SDKs contain code that facilitates building applications that connect to and are managed by Azure IoT Hub services.

Learn more about the IoT Hub Device SDKs in the [IoT Device Development Documentation](#).

OS platform and hardware compatibility

Supported platforms for the SDKs can be found in [Azure IoT SDKs Platform Support](#).

For more information about SDK compatibility with specific hardware devices, see the [Azure Certified for IoT device catalog](#) or individual repository.

NOTE

Some of the features mentioned in this article, like cloud-to-device messaging, device twins, and device management, are only available in the standard tier of IoT Hub. For more information about the basic and standard IoT Hub tiers, see [How to choose the right IoT Hub tier](#).

Next steps

Relevant docs related to development using the Azure IoT SDKs:

- Learn about [how to manage connectivity and reliable messaging](#) using the IoT Hub SDKs.
- Learn about how to [develop for mobile platforms](#) such as iOS and Android.
- [Azure IoT SDK platform support](#)

Other reference topics in this IoT Hub developer guide include:

- [IoT Hub endpoints](#)
- [IoT Hub query language for device twins, jobs, and message routing](#)
- [Quotas and throttling](#)
- [IoT Hub MQTT support](#)
- [IoT Hub REST API reference](#)

Azure IoT Hub SDKs

6/8/2021 • 2 minutes to read • [Edit Online](#)

There are two categories of software development kits (SDKs) for working with IoT Hub:

- **IoT Hub Service SDKs** enable you to build backend applications to manage your IoT hub, and optionally send messages, schedule jobs, invoke direct methods, or send desired property updates to your IoT devices or modules.
- **IoT Hub Device SDKs** enable you to build apps that run on your IoT devices using device client or module client. These apps send telemetry to your IoT hub, and optionally receive messages, job, method, or twin updates from your IoT hub. You can use these SDKs to build device apps that use [Azure IoT Plug and Play](#) conventions and models to advertise their capabilities to IoT Plug and Play-enabled applications. You can also use module client to author [modules](#) for [Azure IoT Edge runtime](#).

In addition, we also provide a set of SDKs for working with the [Device Provisioning Service](#).

- **Provisioning Device SDKs** enable you to build apps that run on your IoT devices to communicate with the Device Provisioning Service.
- **Provisioning Service SDKs** enable you to build backend applications to manage your enrollments in the Device Provisioning Service.

Learn about the [benefits of developing using Azure IoT SDKs](#).

Azure IoT Hub Service SDKs

The Azure IoT service SDKs contain code to facilitate building applications that interact directly with IoT Hub to manage devices and security.

PLATFORM	PACKAGE	CODE REPOSITORY	SAMPLES	REFERENCE
.NET	NuGet	GitHub	Samples	Reference
Java	Maven	GitHub	Samples	Reference
Node	npm	GitHub	Samples	Reference
Python	pip	GitHub	Samples	Reference
Node.js	npm	GitHub	Samples	Reference

Azure IoT Hub service SDK for iOS:

- Install from [CocoaPod](#)
- [Samples](#)

Microsoft Azure Provisioning SDKs

The [Microsoft Azure Provisioning SDKs](#) enable you to provision devices to your IoT Hub using the [Device Provisioning Service](#).

Platform	Package	Source Code	Reference
.NET	Device SDK, Service SDK	GitHub	Reference
C	apt-get, MBED, Arduino IDE or iOS	GitHub	Reference
Java	Maven	GitHub	Reference
Node.js	Device SDK, Service SDK	GitHub	Reference
Python	Device SDK, Service SDK	GitHub	Device Reference, Service Reference

Azure IoT Hub Device SDKs

The Microsoft Azure IoT device SDKs contain code that facilitates building applications that connect to and are managed by Azure IoT Hub services.

Learn more about the IoT Hub Device SDKs in the [IoT Device Development Documentation](#).

OS platform and hardware compatibility

Supported platforms for the SDKs can be found in [Azure IoT SDKs Platform Support](#).

For more information about SDK compatibility with specific hardware devices, see the [Azure Certified for IoT device catalog](#) or individual repository.

NOTE

Some of the features mentioned in this article, like cloud-to-device messaging, device twins, and device management, are only available in the standard tier of IoT Hub. For more information about the basic and standard IoT Hub tiers, see [How to choose the right IoT Hub tier](#).

Next steps

Relevant docs related to development using the Azure IoT SDKs:

- Learn about [how to manage connectivity and reliable messaging](#) using the IoT Hub SDKs.
- Learn about how to [develop for mobile platforms](#) such as iOS and Android.
- [Azure IoT SDK platform support](#)

Other reference topics in this IoT Hub developer guide include:

- [IoT Hub endpoints](#)
- [IoT Hub query language for device twins, jobs, and message routing](#)
- [Quotas and throttling](#)
- [IoT Hub MQTT support](#)
- [IoT Hub REST API reference](#)

Azure Maps community - Open-source projects

6/1/2021 • 4 minutes to read • [Edit Online](#)

These open-source, community-driven initiatives are created and maintained by the Azure Maps team. They're not part of the standard product or service offerings.

The following lists some of the most popular Azure Maps open-source projects and samples.

Bots

PROJECT NAME	DESCRIPTION
Bot Framework - Point of Interest skill	The Point of Interest Skill provides POI related capabilities to a Virtual Assistant using Azure Maps with Azure Bot Service and Bot Framework.
BotBuilder Location	An open-source location picker control for Microsoft Bot Framework powered by Bing Maps REST services.

Open Web SDK modules

The following is a list of open-source projects that extend the capabilities of the Azure Maps Web SDK.

PROJECT NAME	DESCRIPTION
Azure Maps Animation module	A rich library of animations for use with the Azure Maps Web SDK.
Azure Maps Bring Data Into View Control module	An Azure Maps Web SDK module that provides a control that makes it easy to bring any data loaded on the map into view.
Azure Maps Geolocation Control module	An Azure Maps Web SDK module that provides a control that uses the browser's geolocation API to locate the user on the map.
Azure Maps Gridded Data Source module	A module for the Azure Maps Web SDK that provides a data source that clusters data points into cells of a grid area. This operation is also known by many names such as tessellations, data binning, or hex bins.
Azure Maps Fullscreen Control module	An Azure Maps Web SDK module that provides a control to display the map in fullscreen mode.
Azure Maps HTML Marker Layer module	An Azure Maps Web SDK module that provides a layer that renders point data from a data source as HTML elements on the map.
Azure Maps Image Exporter module	A module for the Azure Maps Web SDK that generates screenshots of the map.

PROJECT NAME	DESCRIPTION
Azure Maps Overview Map module	An Azure Maps Web SDK module that provides a control that displays an overview map of the area the main map is focused on.
Azure Maps Scale Bar Control module	An Azure Maps Web SDK module that provides a control that displays a scale bar relative to the pixel resolution at the center of the map.
Azure Maps Selection Control module	An Azure Maps Web SDK module that provides controls for selecting data in a data source using drawing tools or by requesting a route range polygon.
Azure Maps Services UI module	A set of web UI controls that wrap the Azure Maps REST services.
Azure Maps Spider Clusters module	A module for the Azure Maps Web SDK that adds a visualization to the map which expands clusters into a spiral spider layout.
Azure Maps Spyglass Control module	An Azure Maps Web SDK module that provides a window that displays a data set inside of a spyglass on the map.
Azure Maps Swipe Map module	A module for the Azure Maps Web SDK that allows swiping between two overlapping maps, ideal for comparing two overlapping data sets.
Azure Maps Sync Map module	An Azure Maps Web SDK module that synchronizes the cameras of two or more maps.

Samples

PROJECT NAME	DESCRIPTION
Azure Maps Code Samples	A collection of code samples for using Azure Maps in web-based apps.
Azure Maps Gov Cloud Code Samples	A collection of code samples for using Azure Maps through Azure Government Cloud.
Azure Maps & Azure Active Directory Samples	A collection of samples that show how to use Azure Active Directory with Azure Maps.
LiveMaps	Sample application to provide live indoor maps visualization of IoT data on top of Azure Maps using Azure Maps Creator
Azure Maps Jupyter Notebook samples	A collection of python samples using the Azure Maps REST services.
Azure Maps .NET UWP IoT Remote Control	This is a sample application that shows how to build a remotely controlled map using Azure Maps and IoT hub services.
Implement IoT spatial analytics using Azure Maps	Tracking and capturing relevant events that occur in space and time is a common IoT scenario.

Third party map control plugins

PROJECT NAME	DESCRIPTION
Azure Maps Cesium plugin	A Cesium JS plugin that makes it easy to integrate Azure Maps services such as tile layers and geocoding services .
Azure Maps Leaflet plugin	A leaflet JavaScript plugin that makes it easy to overlay tile layers from the Azure Maps tile services .
Azure Maps OpenLayers plugin	A OpenLayers JavaScript plugin that makes it easy to overlay tile layers from the Azure Maps tile services .

Tools and resources

PROJECT NAME	DESCRIPTION
Azure Maps Docs	Source for all Azure Location Based Services documentation.
Azure Maps Creator Tools	Python tools for Azure Maps Creator Tools.

A longer list of open-source projects for Azure Maps that includes community created projects is available [here](#)

Supportability of open-source projects

The following statements apply across all of the Azure Maps created and maintained open-source projects and samples:

- Azure Maps open-source projects and samples are created by Microsoft and the community.
- Azure Maps open-source projects and samples are maintained by Microsoft and the community.
- Azure Maps open-source projects and samples use supported and recommended techniques.
- Azure Maps open-source projects and samples are a community initiative – people who work on the initiative for the benefit of others, and have their normal day job as well.
- Azure Maps open-source projects and samples are NOT a product, and therefore it's not supported by Premier Support or other official support channels.
- Azure Maps open-source projects and samples are supported in similar ways as other open-source projects done by Microsoft with support from the community by the community.

Next steps

Find more open-source Azure Maps projects.

[Code samples](#)

Glossary

11/2/2020 • 22 minutes to read • [Edit Online](#)

The following list describes common words used with the Azure Maps services.

A

Address validation: The process of verifying the existence of an address.

Advanced routing: A collection of services that perform advance operations using road routing data; such as, calculating reachable ranges (isochrones), distance matrices, and batch route requests.

Aerial imagery: See [Satellite imagery](#).

Along a route search: A spatial query that looks for data within a specified detour time or distance from a route path.

Altitude: The height or vertical elevation of a point above a reference surface. Altitude measurements are based on a given reference datum, such as mean sea level. See also elevation.

Ambiguous: A state of uncertainty in data classification that exists when an object may appropriately be assigned two or more values for a given attribute. For example, when geocoding "CA", two ambiguous results are returned: "Canada" and "California". "CA" is a country/region and a state code, for "Canada" and "California", respectively.

Annotation: Text or graphics displayed on the map to provide information to the user. Annotation may identify or describe a specific map entity, provide general information about an area on the map, or supply information about the map itself.

Antimeridian: Also known as the 180th Meridian. This is the point where -180 degrees and 180 degrees of longitude meet. Which is the opposite of the prime meridian on the globe.

Application Programming Interface (API): A specification that allows developers to create applications.

API key: See [Shared key authentication](#).

Area of Interest (AOI): The extent used to define a focus area for either a map or a database production.

Asset tracking: The process of tracking the location of an asset, such as a person, vehicle, or some other object.

Asynchronous request: An HTTP request that opens a connection and makes a request to the server that returns an identifier for the asynchronous request, then closes the connection. The server continues to process the request and the user can check the status using the identifier. When the request is finished processing, the user can then download the response. This type of request is used for long running processes.

Autocomplete: A feature in an application that predicts the rest of a word a user is typing.

Autosuggest: A feature in an application that predicts logical possibilities for what the user is typing.

Azure Location Based Services (LBS): The former name of Azure Maps when it was in preview.

Azure Active Directory (Azure AD): Azure AD is Microsoft's cloud-based identity and access management service. Azure Maps Azure AD integration is currently available in preview for all Azure Maps APIs. Azure AD supports Azure role-based access control (Azure RBAC) to allow fine-grained access to Azure Maps resources. To learn more about Azure Maps Azure AD integration, see [Azure Maps and Azure AD](#) and [Manage authentication in Azure Maps](#).

Azure Maps key: See [Shared key authentication](#).

B

Base map: The part of a map application that displays background reference information such as roads, landmarks, and political boundaries.

Batch request: The process of combining multiple requests into a single request.

Bearing: The horizontal direction of a point in relation to another point. This is expressed as an angle relative to north, from 0-degrees to 360 degrees in a clockwise direction.

Boundary: A line or polygon separating adjacent political entities, such as countries/regions, districts, and properties. A boundary is a line that may or may not follow physical features, such as rivers, mountains, or walls.

Bounds: See [Bounding box](#).

Bounding box: A set of coordinates used to represent a rectangular area on the map.

C

Cadastre: A record of registered land and properties. See also [Parcel](#).

Camera: In the context of an interactive map control, a camera defines the maps field of view. The viewport of the camera is determined based on several map parameters: center, zoom level, pitch, bearing.

Centroid: The geometric center of a feature. The centroid of a line would be the midpoint while the centroid of a polygon would be its center of area.

Choropleth map: A thematic map in which areas are shaded in proportion to a measurement of a statistical variable. This statistical variable is displayed on the map. For example, coloring the boundary of each US state based on its relative population to all other states.

Concave hull: A shape that represents a possible concave geometry that encloses all shapes in the specified data set. The generated shape is similar to wrapping the data with plastic wrap and then heating it, thus causing large spans between points to cave in towards other data points.

Consumption model: Information that defines the rate at which a vehicle consumes fuel or electricity. Also see the [consumption model documentation](#).

Control: A self-contained or reusable component consisting of a graphical user interface that defines a set of behaviors for the interface. For example, a map control, is generally the portion of the user interface that loads an interactive map.

Convex hull: A convex hull is a shape that represents the minimum convex geometry that encloses all shapes in the specified data set. The generated shape is similar to wrapping an elastic band around the data set.

Coordinate: Consists of the longitude and latitude values used to represent a location on a map.

Coordinate system: A reference framework used to define the positions of points in space in two or three dimensions.

Country code: A unique identifier for a country/region based on the ISO standard. ISO2 is a two-character code for a country/region (for example, US), which ISO3 represents a three-character code (for example, USA).

Country subdivision: A first-level subdivision of a country/region, commonly known as a state or province.

Country secondary subdivision: A second-level subdivision of a country/region, commonly known as a county.

Country tertiary subdivision: A third-level subdivision of a country/region, typically a named area such as a ward.

Cross street: A point where two or more streets intersect.

Cylindrical projection: A projection that transforms points from a spheroid or sphere onto a tangent or secant cylinder. The cylinder is then sliced from top to bottom and flattened into a plane.

D

Datum: The reference specifications of a measurement system, a system of coordinate positions on a surface (a horizontal datum) or heights above or below a surface (a vertical datum).

DBF file: A database file format that is used in combination with Shapefiles (SHP).

Degree Minutes Seconds (DMS): The unit of measure for describing latitude and longitude. A degree is $1/360^{\text{th}}$ of a circle. A degree is further divided into 60 minutes, and a minute is divided into 60 seconds.

Delaunay triangulation: A technique for creating a mesh of contiguous, nonoverlapping triangles from a dataset of points. Each triangle's circumscribing circle contains no points from the dataset in its interior.

Demographics: The statistical characteristics (such as age, birth rate, and income) of a human population.

Destination: An end point or location in which someone is traveling to.

Digital Elevation Model (DEM): A dataset of elevation values related to a surface, captured over an area in regular intervals using a common datum. DEMs are typically used to represent terrain relief.

Dijkstra's algorithm: An algorithm that examines the connectivity of a network to find the shortest path between two points.

Distance matrix: A matrix that contains travel time and distance information between a set of origins and destinations.

E

Elevation: The vertical distance of a point or an object above or below a reference surface or datum. Generally, the reference surface is mean sea level. Elevation generally refers to the vertical height of land.

Envelope: See [Bounding box](#).

Extended postal code: A postal code that may include additional information. For example, in the USA, zip codes have five digits. But, an extended zip code, known as zip+4, includes four additional digits. These additional digits are used to identify a geographic segment within the five-digit delivery area, such as a city block, a group of apartments, or a post office box. Knowing the geographic segment aids in efficient mail sorting and delivery.

Extent: See [Bounding box](#).

F

Federated authentication: An authentication method that allows a single logon/authentication mechanism to be used across multiple web and mobile apps.

Feature: An object that combines a geometry with an additional metadata information.

Feature collection: A collection of feature objects.

Find along route: A spatial query that looks for data that is within a specified detour time or distance from a route path.

Find nearby: A spatial query that searches a fixed straight-line distance (as the crow flies) from a point.

Fleet management: The management of commercial vehicles such as cars, trucks, ships, and planes. Fleet management can include a range of functions, such as vehicle financing, maintenance, telematics (tracking and diagnostics) as well as driver, speed, fuel, and health and safety management. Fleet Management is a process used by companies who rely on transportation in their business. The companies want to minimize the risks and reduce their overall transportation and staff costs, while ensuring compliance with government legislation.

Free flow speed: The free flow speed expected under ideal conditions. Usually the speed limit.

Free form address: A full address that is represented as a single line of text.

Fuzzy search: A search that takes in a free form string of text that may be an address or point of interest.

G

Geocode: An address or location that has been converted into a coordinate that can be used to display that location on a map.

Geocoding: Also known as forward geocoding, is the process of converting address or location data into coordinates.

Geodesic path: The shortest path between two points on a curved surface. When rendered on Azure Maps this path appears as a curved line due to the Mercator projection.

Geofence: A defined geographical region that can be used to trigger events when a device enters or exists the region.

GeoJSON: Is a common JSON-based file format used for storing geographical vector data such as points, lines, and polygons. **Note:** Azure Maps uses an extended version of GeoJSON as [documented here](#).

Geometry: Represents a spatial object such as a point, line, or polygon.

GeometryCollection: A collection of geometry objects.

GeoPol: Refers to geopolitically sensitive data, such as disputed borders and place names.

Georeference: The process of aligning geographic data or imagery to a known coordinate system. This process may consist of shifting, rotating, scaling, or skewing the data.

GeoRSS: An XML extension for adding spatial data to RSS feeds.

GIS: An acronym for "Geographic Information System". A common term used to describe the mapping industry.

GML: Also known as Geography Markup Language. An XML file extension for storing spatial data.

GPS: Also known as Global Positioning System, is a system of satellites used for determining a devices position on the earth. The orbiting satellites transmit signals that allow a GPS receiver anywhere on earth to calculate its own location through trilateration.

GPX: Also known as GPS eXchange format, is an XML file format commonly created from GPS devices.

Great-circle distance: The shortest distance between two points on the surface of a sphere.

Greenwich Mean Time (GMT): The time at the prime meridian, which runs through the Royal Observatory in Greenwich, England.

GUID: A globally unique identifier. A string used to uniquely identify an interface, class, type library, component category, or record.

H

Haversine formula: A common equation used for calculating the great-circle distance between two points on a sphere.

HD maps: Also known as High Definition Maps, consists of high fidelity road network information such as lane markings, signage, and direction lights required for autonomous driving.

Heading: The direction something is pointing or facing. See also [Bearing](#).

Heatmap: A data visualization in which a range of colors represent the density of points in a particular area. See also Thematic map.

Hybrid imagery: Satellite or aerial imagery that has road data and labels overlaid on top of it.

I

IANA: An acronym for the Internet Assigned Numbers Authority. A nonprofit group that oversees global IP address allocation.

Isochrone: An isochrone defines the area in which someone can travel within a specified time for a mode of transportation in any direction from a given location. See also [Reachable Range](#).

Isodistance: Given a location, an isochrone defines the area in which someone can travel within a specified distance for a mode of transportation in any direction. See also [Reachable Range](#).

K

KML: Also known as Keyhole Markup Language, is a common XML file format for storing geographic vector data such as points, lines, and polygons.

L

Landsat: Multispectral, earth-orbiting satellites developed by NASA that gather imagery of land. This imagery is used in many industries such as agriculture, forestry, and cartography.

Latitude: The angular distance measured in degrees from equator in a north or south direction.

Level of detail: See Zoom level.

Lidar: Acronym for light detection and ranging. A remote-sensing technique that uses lasers to measure distances to reflective surfaces.

Linear interpolation: The estimation of an unknown value using the linear distance between known values.

LineString: A geometry used to represent a line. Also known as a polyline.

Localization: Support for different languages and cultures.

Logistics: The process of moving people, vehicles, supplies, or assets in a coordinated way.

Longitude: The angular distance measured in degrees from the prime meridian in an east or west direction.

M

Map Tile: A rectangular image that represents a partition of a map canvas. For more information, see the [Zoom levels and tile grid documentation](#).

Marker: Also known as a pin or pushpin, is an icon that represents a point location on a map.

Mercator projection: A cylindrical map projection that became the standard map projection for nautical purposes because of its ability to represent lines of constant course, known as rhumb lines, as straight segments that conserve the angles with the meridians. All flat map projections distort the shapes or sizes of the map when compared to the true layout of the Earth's surface. The Mercator projection exaggerates areas far from the equator, such that smaller areas appear larger on the map as you approach the poles.

MultiLineString: A geometry that represents a collection of LineString objects.

MultiPoint: A geometry that represents a collection of Point objects.

MultiPolygon: A geometry that represents a collection of Polygon objects. For example, to show the boundary of Hawaii, each island would be outlined with a polygon. Thus, the boundary of Hawaii would thus be a MultiPolygon.

Municipality: A city or town.

Municipality subdivision: A subdivision of a municipality, such as a neighborhood or local area name such as "downtown".

N

Navigation bar: The set of controls on a map used for adjusting the zoom level, pitch, rotation, and switching the base map layer.

Nearby search: A spatial query that searches a fixed straight-line distance (as the crow flies) from a point.

Neutral Ground Truth: A map that renders labels in the official language of the region it represents and in local scripts if available.

O

Origin: A start point or location in which a user is.

P

Panning: The process of moving the map in any direction while maintaining a constant zoom level.

Parcel: A plot of land or property boundary.

Pitch: The amount of tilt the map has relative to the vertical where 0 is looking straight down at the map.

Point: A geometry that represents a single position on the map.

Points of interest (POI): A business, landmark, or common place of interest.

Polygon: A solid geometry that represents an area on a map.

Polyline: A geometry used to represent a line. Also known as a LineString.

Position: The longitude, latitude, and altitude (x,y,z coordinates) of a point.

Post code: See [Postal code](#).

Postal code: A series of letters or numbers, or both, in a specific format. The postal-code is used by the postal service of a country/region to divide geographic areas into zones in order to simplify delivery of mail.

Primary key: The first of two subscriptions keys provided for Azure Maps shared key authentication. See [Shared key authentication](#).

Prime meridian: A line of longitude that represents 0-degrees longitude. Generally, longitude values decrease

when traveling in a westerly direction until 180 degrees and increase when traveling in easterly directions to -180-degrees.

PRJ: A text file which often accompanies a Shapefile file that contains information about the projected coordinate system the data set is in.

Projection: A projected coordinate system based on a map projection such as transverse Mercator, Albers equal area, and Robinson. These provide the ability to project maps of the earth's spherical surface onto a two-dimensional Cartesian coordinate plane. Projected coordinate systems are sometimes referred to as map projections.

Q

Quadkey: A base-4 address index for a tile within a quadtree tiling system. For more information, see [Zoom levels and tile grid](#) documentation for more information.

Quadtree: A data structure in which each node has exactly four children. The tiling system used in Azure Maps uses a quadtree structure such that as a user zooms in one level, each map tile breaks up into four subtiles. For more information, see [Zoom levels and tile grid](#) documentation for more information.

Queries Per Second (QPS): The number of queries or requests that can be made to a service or platform within one second.

R

Radial search: A spatial query that searches a fixed straight-line distance (as the crow flies) from a point.

Raster data: A matrix of cells (or pixels) organized into rows and columns (or a grid) where each cell contains a value representing information, such as temperature. Raster's include digital aerial photographs, imagery from satellites, digital pictures, and scanned maps.

Raster layer: A tile layer that consists of raster images.

Reachable range: A reachable range defines the area in which someone can travel within a specified time or distance, for a mode of transportation to travel, in any direction from a location. See also [Isochrone](#) and [Isodistance](#).

Remote sensing: The process of collecting and interpreting sensor data from a distance.

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.

Reproject: See [Transformation](#).

REST service: Acronym for Representational State Transfer. An architecture for exchanging information between peers in a decentralized, distributed environment. REST allows programs on different computers to communicate independently of an operating system or platform. A service can send a Hypertext Transfer Protocol (HTTP) request to a uniform resource locator (URL) and get back data.

Route: A path between two or more locations, which may also include additional information such as instructions for waypoints along the route.

Requests Per Second (RPS): See [Queries Per Second \(QPS\)](#).

RSS: Acronym for Really Simple Syndication, Resource Description Framework (RDF) Site Summary, or Rich Site Summary, depending on the source. A simple, structured XML format for sharing content among different Web sites. RSS documents include key metadata elements such as author, date, title, a brief description, and a hypertext link. This information helps a user (or an RSS publisher service) decide what materials are worth further investigation.

S

Satellite imagery: Imagery that has been captured by planes and satellites pointing straight down.

Secondary key: The second of two subscriptions keys provided for Azure Maps shared key authentication. See [Shared key authentication](#).

Shapefile (SHP): Also known as an ESRI Shapefile, is a vector data storage format for storing the location, shape, and attributes of geographic features. A shapefile is stored in a set of related files.

Shared key authentication: Shared Key authentication relies on passing Azure Maps account generated keys with each request to Azure Maps. These keys are often referred to as subscription keys. It is recommended that keys are regularly regenerated for security. Two keys are provided so that you can maintain connections using one key while regenerating the other. When you regenerate your keys, you must update any applications that access this account to use the new keys. To learn more about Azure Maps authentication, see [Azure Maps and Azure AD](#) and [Manage authentication in Azure Maps](#).

Software development kit (SDK): A collection of documentation, sample code, and sample apps to help a developer use an API to build apps.

Spherical Mercator projection: See [Web Mercator](#).

Spatial query: A request made to a service that performs a spatial operation. Such as a radial search, or along a route search.

Spatial reference: A coordinate-based local, regional, or global system used to precisely locate geographical entities. It defines the coordinate system used to relate map coordinates to locations in the real world. Spatial references ensure spatial data from different layers, or sources, can be integrated for accurate viewing or analysis. Azure Maps uses the [EPSG:3857](#) coordinate reference system and WGS 84 for input geometry data.

SQL spatial: Refers to the spatial functionality built into SQL Azure and SQL Server 2008 and above. This spatial functionality is also available as a .NET library that can be used independently of SQL Server. For more information, see the [Spatial Data \(SQL Server\) documentation](#) for more information.

Subscription key: See [Shared key authentication](#).

Synchronous request: An HTTP request opens a connection and waits for a response. Browsers limit the number of concurrent HTTP requests that can be made from a page. If multiple long running synchronous requests are made at the same time, then this limit can be reached. Requests will be delayed until one of the other requests has completed.

T

Telematics: Sending, receiving, and storing information via telecommunication devices in conjunction with effecting control on remote objects.

Temporal data: Data that specifically refers to times or dates. Temporal data may refer to discrete events, such as lightning strikes; moving objects, such as trains; or repeated observations, such as counts from traffic sensors.

Terrain: An area of land having a particular characteristic, such as sandy terrain or mountainous terrain.

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/regions based on some metric of data.

Tile layer: A layer displayed by assembling map tiles (rectangular sections) into a continuous layer. The tiles are either raster image tiles or vector tiles. Raster tile layers are typically rendered ahead of time, and they're stored as images on a server. Raster tile layers may use a large storage space. Vector tile layers are rendered near real time within the client application. Thus, the server-side storage requirements are smaller for vector tile layers.

Time zone: A region of the globe that observes a uniform standard time for legal, commercial, and social purposes. Time zones tend to follow the boundaries of countries/regions and their subdivisions.

Transaction: Azure Maps uses a transactional licensing model where;

- One transaction is created for every 15 map or traffic tiles requested.
- One transaction is created for each API call to one of the services in Azure Maps. Searching and routing are examples of Azure Maps service.

Transformation: The process of converting data between different geographic coordinate systems. You may, for example, have some data that was captured in the United Kingdom and based on the OSGB 1936 geographic coordinate system. Azure Maps uses the [EPSG:3857](#) coordinate reference system variant of WGS84. As such to display the data correctly, it will need to have its coordinates transformed from one system to another.

Traveling Salesmen Problem (TSP): A Hamiltonian circuit problem in which a salesperson must find the most efficient way to visit a series of stops, then return to the starting location.

Trilateration: The process of determining the position of a point on the earth's surface, with respect to two other points, by measuring the distances between all three points.

Turn-by-turn navigation: An application that provides route instructions for each step of a route as the users approaches the next maneuver.

V

Vector data: Coordinate based data that is represented as points, lines, or polygons.

Vector tile: An open data specification for storing geospatial vector data using the same tile system as the map control. See also [Tile layer](#).

Vehicle Routing Problem (VRP): A class of problems, in which a set of ordered routes for a fleet of vehicles is calculated while taking into consideration as set of constraints. These constraints may include delivery time windows, multiple route capacities, and travel duration constraints.

Voronoi diagram: A partition of space into areas, or cells, that surround a set of geometric objects, usually point features. These cells, or polygons, must satisfy the criteria for Delaunay triangles. All locations within an area are closer to the object it surrounds than to any other object in the set. Voronoi diagrams are often used to delineate areas of influence around geographic features.

W

Waypoint: A waypoint is a specified geographical location defined by longitude and latitude that is used for navigational purposes. Often used to represent a point in which someone navigates a route through.

Waypoint optimization: The process of reordering a set of waypoints to minimize the travel time or distance required to pass through all provided waypoints. Depending on the complexity of the optimization, this optimization is often referred to as the [Traveling Salesmen Problem](#) or [Vehicle Routing Problem](#).

Web Map Service (WMS): WMS is an Open Geographic Consortium (OGC) standard that defines image-based map services. WMS services provide map images for specific areas within a map on demand. Images

include pre-rendered symbology and may be rendered in one of several named styles if defined by the service.

Web Mercator: Also known as Spherical Mercator projection. It's a slight variant of the Mercator projection, one used primarily in Web-based mapping programs. It uses the same formulas as the standard Mercator projection as used for small-scale maps. However, the Web Mercator uses the spherical formulas at all scales, but large-scale Mercator maps normally use the ellipsoidal form of the projection. The discrepancy is imperceptible at the global scale, but it causes maps of local areas to deviate slightly from true ellipsoidal Mercator maps, at the same scale.

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. Azure Maps uses the [EPSG:3857](#) coordinate reference system variant of WGS84.

Z

Z-coordinate: See [Altitude](#).

Zip code: See [Postal code](#).

Zoom level: Specifies the level of detail and how much of the map is visible. When zoomed all the way to level 0, the full world map will often be visible. But, the map will show limited details such as country/region names, borders, and ocean names. When zoomed in closer to level 17, the map will display an area of a few city blocks with detailed road information. In Azure Maps, the highest zoom level is 22. For more information, see the [Zoom levels and tile grid](#) documentation.