Indoor map data management in Azure Maps.| Microsoft Docs

farah-alyasari

# Indoor map data management

Azure Maps provides the Indoor Maps SDK. The SDK is composed of Azure Maps API services, the Indoor Maps module, and a plug-in for the QGIS application. This article provides an overview of the data management pipeline for the Indoor Maps SDK. The process starts with data ingestion into Azure Maps resources, using the Azure Maps API services. Throughout the development process, the Azure Maps APIs can be used for heavy data modification, such as uploading data or importing bulk data to uploaded data. The Azure Maps QGIS plug-in can be used for light data modification using a friendly interface. The Indoor Maps module can be used to render indoor maps and control indoor maps events for a web application. After the development phase completes, data resource can be administrated using the Azure Maps QGIS plug-in or the Azure Maps APIs. The Azure Maps QGIS plug-in is part of the “Private Atlas - Private Preview” tool in the QGIS application. So, you’ll completely own your map data and the rendered maps.

## Prerequisites

You need to [obtain a primary subscription key](quick-demo-map-app.md#get-the-primary-key-for-your-account) by [making an Azure Maps account](quick-demo-map-app.md#create-an-account-with-azure-maps). The primary subscription key is a required parameter for all Azure Maps APIs.

In this article, the [Postman](https://www.postman.com/) application is used to call the Azure Maps APIs, but you may use any API development environment.

Before you feed data to the Azure Maps resources, make sure that your DWG package adheres to the [DWG package requirements](dwg-requirements.md).

## Data Ingestion

Data ingestion begins by using the Data Upload API to upload your DWG package to Azure Maps resources. The API returns a resource URL pointing to the uploaded package. You can use the resource URL to check the upload data status. Once the package is successfully uploaded, you can use the URL to access and download the package. The URL also contains the udid for the uploaded package.

You need the package udid as an input to the Conversion API. This API accesses the uploaded DWG package and validates it against the [DWG package requirements](dwg-requirements.md). If the package meets the requirements, it’s converted from design data to map data. Then, a resource URL is returned. The resource URL contains the udid for the converted package. Once the data is successfully uploaded and converted, the map data can be accessed by other APIs using the conversion ID.

The Conversion API will return error codes in the response body if the DWG package doesn’t meet the requirements. If you receive errors, then see Conversion API error codes for details.

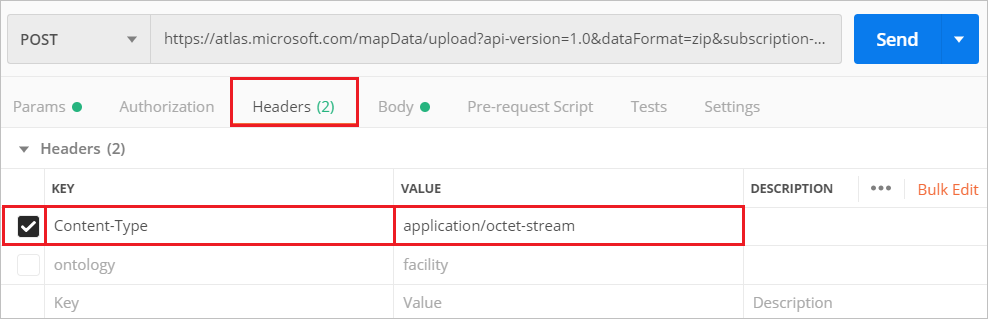
### Data upload procedure

Follow the processes below to upload your DWG design package:

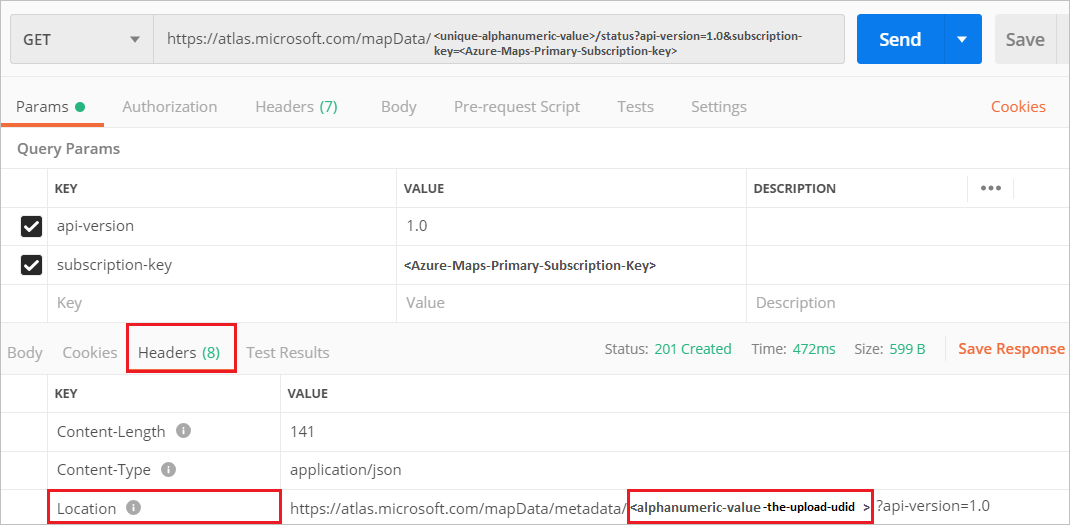
1. Make a **POST** HTTP request using the following URL to upload your DWG package to the Azure Maps service. For this request, and other requests mentioned in this article, replace <Azure-Maps-Primary-Subscription-key> with your primary subscription key.

* https://atlas.microsoft.com/mapData/upload?api-version=1.0&dataFormat=zip&subscription-key=<Azure-Maps-Primary-Subscription-key>

1. In the **Headers**, specify a value for the Content-Type key. In this case, use application/octet-stream because our DWG package is in a zip folder. In the request **Body**, select **binary**. Click on **Select File** and choose your DWG package.

* 
* data-management

1. Once the request completes, go to the **Headers** for the response. Copy the resource URL, it’s the value for the **Location** key. This URL points to the Azure Maps resource that contains the DWG package.
2. Obtain the udid for the uploaded package by making a **GET** HTTP request at the resource URL. You’ll need to append your primary subscription key to the URL, for authentication.
3. When the **GET** HTTP request completes successfully, the response header will contain the udid for the uploaded DWG package. Copy the udid, it’s highlighted in this image, if you’re using the Postman application.

* 
* data-management

### Data conversion process

Now that your DWG package is uploaded, we’ll use the udid for the uploaded package to convert the DWG package into map data.

1. Make a **POST** HTTP request using the following URL to convert your uploaded DWG package into map data. Use the udid for the uploaded DWG package as a parameter for this request.

* https://atlas.microsoft.com/conversion/convert?subscription-key=<Azure-Maps-Primary-Subscription-key>&api-version=1.0&udid=<upload-udid>&inputType=DWG

1. When the request completes, you’ll receive a response header containing a **Location** key. The **Location** key returned by the conversion API holds the resource URL of the converted map data. Copy the resource URL.
2. Make a **GET** HTTP request at the resource URL and append your Azure Maps primary subscription key.
3. Once the request completes successfully, you’ll see a success status message in the response body. Copy the udid for the converted map data from the resourceLocation element. We’ll refer to this udid as the conversion ID, and we’ll use it in the data curation stage.

## Data Curation

The way the map data is managed depends on your application. The Azure Maps Dataset, Tileset, and Feature Stateset services will regularly be used to manage your data. The Data set service lets you create, list, and delete data sets. A data set is the primary resource for compiling one or more facility data. You can also import more data into a data set after it has been uploaded. A tile set is a set of gridded vector tiles. A state set contains state information for map feature properties that can be dynamically manipulated. The tile set and state set services let you create, list, and delete a tile set or a state set, respectively.

The tile set and the state set services use data from the data set service. However, the tile set and state set are independent entities. In other words, if you delete the data set that you used to generate the tile set with, then the tile set will continue to exist. Normally, the tile sets and state sets are used in combination with the Indoor Maps module to render tiles and feature states on the map.

If you want to show dynamic map styling by using the Feature Stateset service, then you’ll need to use the Azure Maps WFS service. The service lets you learn about your map feature collections and features. You need to use this service to obtain a feature ID to use for the Feature state set service.

The next sections detail the process of using the Azure Maps Dataset, Tileset, and Feature Stateset services. In the state sets section, the steps demonstrate how to use Azure Maps WFS service to learn about your map feature. The Azure Maps QGIS plug-in section and the Indoor Module section provide an overview of each tool. We also linked tutorials and how-to guides at the end of this article, to show you how to use each service and tool in more details.

### Data sets

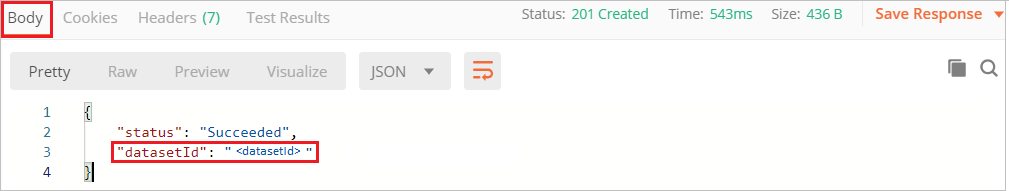
The data set is a collection of map data entities, such as buildings. To make a data set, use the Dataset Create API. It takes a conversion ID for a DWG package, and it produces a data set with a unique data set ID.

The steps below show you how to make a data set, and obtain its ID:

1. Make a **POST** request to the Dataset Create API to create a new data set. The URL of the request should have a format like the one below:

* https://atlas.microsoft.com/dataset/create?api-version=1.0&conversionID=<your-dwg-package-conversion-ID>&type=facility&subscription-key=<Azure-Maps-Primary-Subscription-key>

1. Obtain the resource URL in the **Location** key of the response **Headers**.
2. Make a **GET** request at the URL to Obtain the data set ID. As usual, append your Azure Maps primary subscription key for authentication.
3. Copy the datasetId in the response body.

* 
* data-management

You can use your data set ID for various purposes. For example, you can use the [Azure Maps QGIS plug-in](azure-maps-qgis-plugin.md) to edit the data set, and you would accesses the data set using the data set ID. You may also query your data set by using the Azure Maps Web Feature Service (WFS) API. It’s okay to use another WFS API, as long as it follows the [OGC Web Feature Service 3.0](http://docs.opengeospatial.org/DRAFTS/17-069.html)

You can also generate a tile set and state set from the data in the data set. You can [use the Indoor Maps module](how-to-use-indoor-module.md) to render your vector tiles or feature states. And at any time, you may use the Dataset List API to view details of your data sets. When you’re done using a data set, you can remove it using the Dataset Delete API.

[!Note] The Dataset Create API doesn’t prevent you from appending duplicate blob into a data set.

### Tile sets

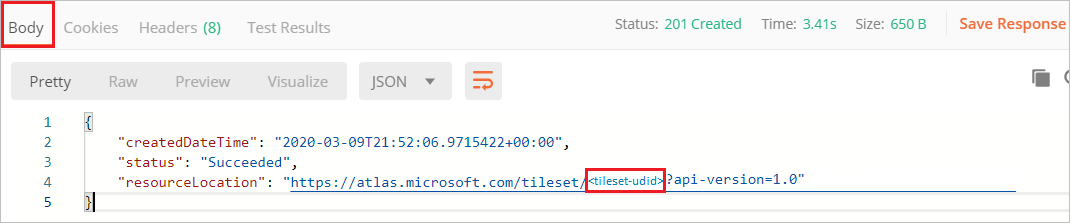
A tile set is a set of vector tiles to render on the map. It’s produced from the data in a data set, so, the Tileset Create API requires a data set ID. And although the tile set is built from the data in the data set, it’s an independent entity. If the data set is deleted, the tile set will continue to exist.

To create a tile set, follow the steps below:

1. Make a **POST** request using the following URL:

* https://atlas.microsoft.com/tileset/create/vector?api-version=1.0&datasetID=<datasetID>&subscription-key=<Azure-Maps-Primary-Subscription-key>

1. The response headers will contain a location header with the tile set resource URL. Make a **GET** request at the URL, with your Azure Maps subscription appended to the end for authentication.
2. Upon a successful request, you’ll receive the udid for the created tile set in the response body. This is also referred to as the tile set ID.

* 
* data-management

To display the map data that was originally obtained from the DWG package, [use the Indoor Maps module](how-to-use-indoor-module.md#use-the-indoor-maps-module) with your tile set ID. You may render indoor maps using the Get Map Tile API. You can also fetch a list of all the tile sets, including the tile sets that aren’t used in your application, using the Tileset List API. The API returns a list of details for your tiles, such as the bounding box.

To delete your tile set, use the Tileset Delete API. If you delete an in-use tile set, then those tiles won’t render on the map at the application runtime.

[!Caution] Whenever you review the list of items and decide to delete them, consider the cascading dependencies impact. For example, you may have a tile set being rendered in your application using the **Get Map Tile API** and deleting the tile set will result in failure to render that tile set.

### Feature State sets

A state set contains style information of properties that can dynamically change. Accessing and changing the style of a feature is useful for highlighting events or states. For example, you may toggle the tile colors to indicate the room occupancy status.

The process of using the feature state set APIs is like using the data set and tile set APIs. First, make a state set. Then, use the feature ID and the state set ID to update the feature properties. The steps below show you how to obtain the feature ID. And, how to create a state set, update the feature ID state, and acquire the state of the feature.

1. If you don’t already have a feature ID, obtain a feature ID from a data set using the WFS API. Make a **GET** request to learn about the collections in your data set.

* https://atlas.microsoft.com/wfs/datasets/<datasetID>/collections?subscription-key=<Azure-Maps-Primary-Subscription-key>&api-version=1.0

1. The response body will contain a set of feature collections in your data set. Once you choose a collection, make a **GET** request at the collection URL to learn about its features. The URL format would look like the one below:

* https://atlas.microsoft.com/wfs/datasets/<datasetID>/collections/unit/items?subscription-key=<Azure-Maps-Primary-Subscription-key>&api-version=1.0

1. Choose and copy a feature **ID** from the response body. We’ll refer to the dynamic properties of this feature as states, and we’ll use the feature ID to modify its states.
2. At this point, we have a feature ID and data set ID. To make a state set, make a **POST** request to the Create Stateset API. Use the data set ID of the data set that contains the state you want to modify. Here’s how the URL should look like:

* https://atlas.microsoft.com/featureState/stateset?api-version=1.0&datasetID=<datasetID>&subscription-key=<Azure-Maps-Primary-Subscription-key>

1. Copy the state set ID from the response body.
2. Use the Feature Update States API to update the state. Pass the state set ID, data set ID, and feature ID, with your Azure Maps subscription key. Here’s a URL of a **POST** request to update the state:

* https://atlas.microsoft.com/featureState/state?api-version=1.0&statesetID=<statesetID>&datasetID=<datasetID>&featureID=<featureID>&subscription-key=<Azure-Maps-Primary-Subscription-key>

1. In the **Headers** of the **POST** request, set Content-Type to application/json. In the **body** of the **POST** request, write the JSON with the feature updates. The updates will only occur if the time posted stamp is after the time stamp of the previous request. Here’s a sample JSON body that updates a feature:

* {  
   "states": [  
   {  
   "keyName": "occupied",  
   "value": true,  
   "eventTimestamp": "2019-11-14T17:10:20"  
   }  
   ]  
  }

1. You may use the Feature Get States API to obtain the status of the state. The URL below shows how the **GET** request would look like:

* https://atlas.microsoft.com/featureState/state?api-version=1.0&statesetID=<statesetID>&datasetID=<datasetID>&featureID=<featureID>&subscription-key=<Azure-Maps-Primary-Subscription-key>

All the dynamic styling properties of all features can become states. For example, the color of a feature within a tile can be a state. One use case, would be to change the color based on room occupancy limit. Another use case would be to toggle the color based on room availability, for a scheduling application. For more information, see Implement dynamic styling for Private Atlas Indoor Maps

The Feature Get States API lets you learn about the state of a feature, using its feature ID. You can render the states on the indoor map using the Indoor Maps manager from the Indoor Module or the Azure Maps QGIS plug-in. You can also delete the state set and its resource using the Feature State Delete API.

[!Caution] Whenever you review the list of state sets and decide to delete them, consider the cascading dependencies impact. For example, if you’re rendering your state set using the Indoor Maps module, deleting a state set will result in failure to render that state set.

### Azure Maps QGIS Plug-in

The Azure Maps QGIS Plug-in lets you make the same data map manipulation as the Azure Maps APIs. However, it’s recommended that you use the plug-in for minor data modification and use the APIs for heavy revisions. In the main window of the plug-in, the user can see the indoor map rendered as it would render in the application. Changes to the data set are observed right after the modification, in near real-time.

The plug-in is still in the experimentation phase. So, there are some limits to keep in mind. Only one user at a time can make changes to the data set. And before switching the floor number, make sure you save your edits for the current floor. Changes done on a given floor will be lost if you don’t save before changing the floor. For more information, see the [Azure Maps QGIS Plug-in](azure-maps-qgis-plugin.md).

### Indoor Maps Module

The Azure Maps Web SDK provides the Indoor Maps module. The Indoor Maps module is an extension to the Map Control module, with functionalities tailored to indoor maps development. It offers a control for the indoor map floor level. The user can attach listener events to execute when a floor level changes, or when the facility changes. To use the Azure Maps module, you would need to obtain the tile set ID to render tiles on the maps. You also render feature states using the state set ID. Both IDs must be assigned in the Indoor Maps manager. Overall, the Azure Maps web SDK and the Indoor Maps module make it convenient to integrate indoor maps with web applications. To get started with the module, read [How to use the Indoor Maps module](how-to-use-indoor-module.md) and the Indoor Module SDK documentation.

## Resource Administration

At this phase, your tile sets are ready for indoor maps rendering. Your data sets can be queried using a WFS service. And your feature state service can accept dynamic property updates. You can continue to monitor the status of your map data using the list APIs available in the Dataset, Tileset, and Feature State services. You can review the relevance of your map data. If you wish, you may update or delete the data using the update and delete APIs, for the respective service.

The Azure Maps QGIS plug-in is another convenient tool to administer your map data. The plug-in interface lets you see real-time updates reflected based on changes to your data set. You can also make minor modification using a graphical user interface rather than making API calls.

## Next steps

Learn more about the different APIs mentioned in this application:

[!div class=“nextstepaction”] Data Upload

[!div class=“nextstepaction”] Data Conversion

[!div class=“nextstepaction”] Dataset

[!div class=“nextstepaction”] Tileset

[!div class=“nextstepaction”] WFS service

[!div class=“nextstepaction”] Feature State set

Relevant articles:

[!div class=“nextstepaction”] [Azure Maps QGIS Plug-in](azure-maps-qgis-plugin.md)

[!div class=“nextstepaction”] Implement dynamic styling for Private Atlas Indoor Maps

[!div class=“nextstepaction”] [Use the Indoor Maps module](how-to-use-indoor-module.md)

[!div class=“nextstepaction”] [Bulk import data to an existing data set](how-to-ingest-bulk-data-in-dataset.md)

[!div class=“nextstepaction”] Indoor Module SDK documentation

Consider leveraging more Azure Maps services with your indoor map application. For example, if your application is “smart”, then you may incorporate IoT system powering location logic or produce map visualization. Learn more by reading: