Indoor map data management in Azure Maps.| Microsoft Docs

# Indoor map data management

Azure Maps provides the Indoor Maps SDK, which has a plug-in for the QGIS application. The Azure Maps QGIS plug-in and the Azure Maps APIs can be used together to develop indoor maps for your application. This article overview the data management pipeline for developing indoor map applications. The process starts with data ingestion into Azure Maps resources, and wraps up with resource administration using the Azure Maps QGIS plug-in.

## 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, including the indoor 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

The first step is to upload your DWG package using the Data Upload API. The Data Upload API consumes a DWG package, and it returns a URI pointing to the uploaded package. Once the package is uploaded, you can use the URI to access and download the package. The URI also contains the udid of the uploaded package. Use 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 URI is returned. The URI contains the conversion udid, and obtaining this id concludes the data ingestion phase. The map data can be accessed using the conversion ID. The data can then be accessed and modified by other APIs.

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

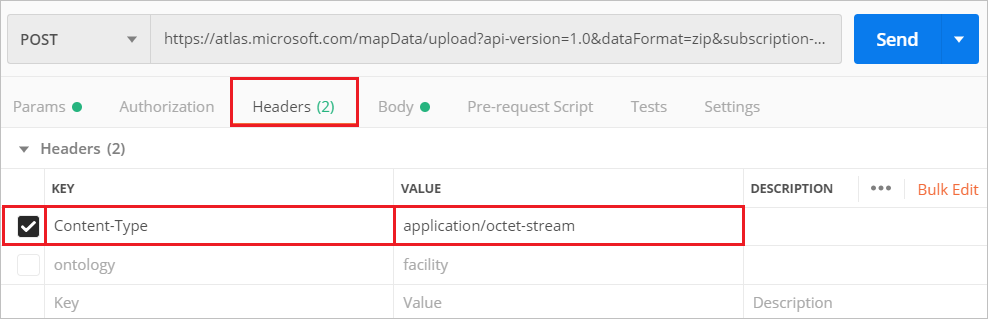
### Data upload process

Follow the processes below to upload your DWG design package:

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.
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 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** tab, specify a value for the Content-Type key, as in the image below. In this case, our DWG package type is application/octet-stream. In the **Body** tab, select **binary**. Click on **Select File** and choose your DWG package.

* 
* data-management

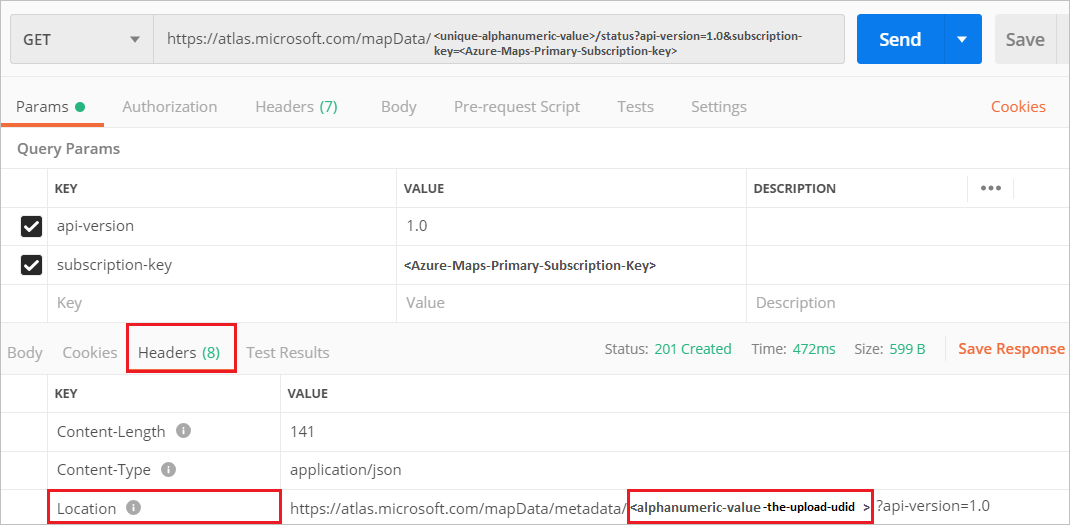
1. Click the blue **Send** button and wait for the request to process. Once the request completes, go to the **Headers** tab for the response, and copy the **Location** URL value. This value points to the Azure Maps resource that contains the DWG package, and it looks like this:

* https://atlas.microsoft.com/mapData/<unique-alphanumeric-value>/status?api-version=1.0

1. Obtain the package ID, or the package udid, by making a **GET** HTTP request at the **Location** URL. You’ll need to append your primary subscription key to the URL, as seen below:

* https://atlas.microsoft.com/mapData/<unique-alphanumeric-value>/status?api-version=1.0&subscription-key=<Azure-Maps-Primary-Subscription-key>

1. When the **GET** HTTP request completes successfully, the response header will contain the udid of the uploaded DWG package. Copy the udid, it’s highlighted in this image.

* 
* data-management

### Data conversion process

keep the Postman application open. Now that your DWG package is uploaded, we’ll use the package udid, from the previous section, to convert the DWG package into map data.

1. Select **New** again. In the **Create New** window, select **Request**. Enter a **Request name**, and select a collection. Click **Save**.
2. Select the **POST** HTTP method in the builder tab and enter the following URL to convert your uploaded DWG package into map data. Use the udid of the uploaded DWG package.

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

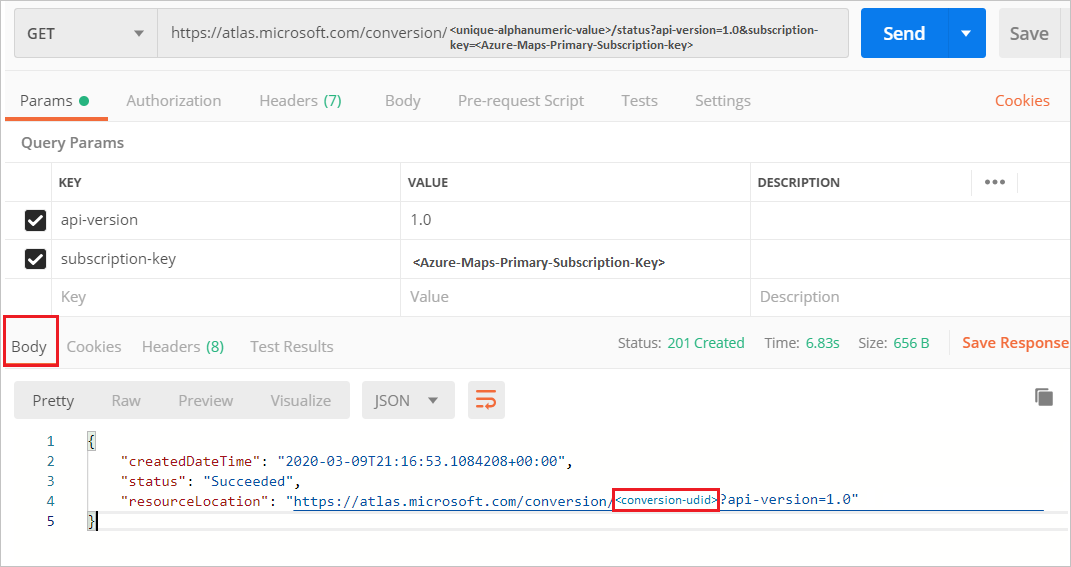
1. When the request completes, you’ll receive a response header containing a **Location** key. The **Location** key returned by the conversion API holds the URL of the converted map data. Copy the location URL, it would have a similar format as the URL below:

* https://atlas.microsoft.com/conversion/<unique-alphanumeric-value>/status?api-version=1.0

1. Start a new **GET** HTTP method in the builder tab. Make a **GET** request at the location URL from the previous step, and append your Azure Maps primary subscription key.

* https://atlas.microsoft.com/conversion/<unique-alphanumeric-value>/status?api-version=1.0&subscription-key=<Azure-Maps-Primary-Subscription-key>

1. Once the request completes successfully, you’ll see a success status message in the response body. Copy the udid of the converted map data from the “resourceLocation”. Or, copy it from the **Headers** response tab. We’ll refer to this udid as the conversion ID for short, and use we’ll it in the data curation stage.

* 
* data-management

[!Note] Sometimes it’ll take a while to show the response body of the API call. If you’re using the postman application and it takes a long time for the response status to change, then click on the send button again.

## Data Curation

In the data curation stage, the map data is maintained and managed. And the way the map data is managed depends on your application. Nonetheless, there are three sets of APIs that will regularly be used at this stage. These APIs are the data set APIs, tile set APIs, and state set APIs. A data set is the primary resource for compiling one or more facility data. A tile set is a set of gridded vector tiles. And, a state set contains state information for the dynamic map properties. The tile set and the state set use data from the data set. So, you would first generate a data set, and then use the data set to generate tile and state sets. Tile sets and state sets are used in combination with the indoor map sdk, to develop your application’s use cases.

The next sections detail the process to generate data sets, tile sets, and state sets. Although the sections focus on some APIs more than others, the general concepts apply to all APIs. You can also use the Azure Maps QGIS plug-in to manipulate the map data. But, you would still need to obtain the set IDs by making the create API calls.

### Data sets

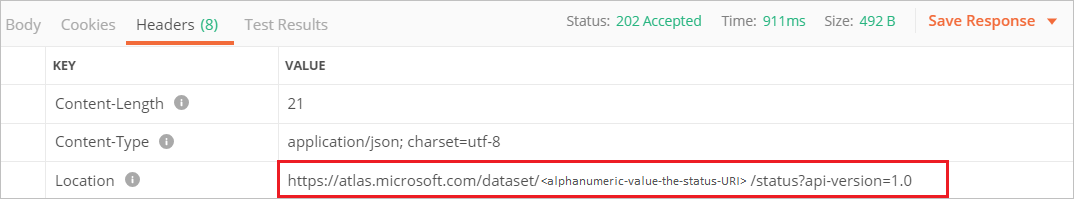
The data set is a collection of map data entities, such as buildings. The Dataset Create API takes a conversion ID of a DWG package, and produces one or more data sets. It assigns a unique data set ID for the newly generated set. It’s not likely that you want to have set duplicates. However, the API doesn’t prevent you from appending duplicate blob into a data set. This option makes the API flexible to meet the various need of developers.

Let’s now make a new data set using the Dataset Create API and the conversion ID of our DWG package, obtained in the data ingestion phase.

1. Open the Postman application. Select **New**. In the **Create New** window, select **Request**. Enter a **Request name**, and select a collection. Click **Save**
2. 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>

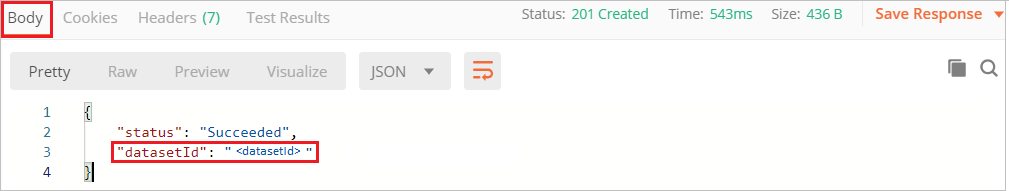
1. Obtain the URL in the **Location** key of the response **Headers** tab.

* 
* data-management

1. Make a **GET** request at the URL to Obtain the data set ID. As usual, append your Azure Maps primary subscription key for authentication.

* https://atlas.microsoft.com/dataset/<alphanumeric-value-status-URI>/status?api-version=1.0&subscription-key=<Azure-Maps-Primary-Subscription-key>

1. Copy the datasetID in the response body.

* 
* data-management

Now that you’ve obtained the data set ID, you can use the [Azure Maps QGIS plug-in](azure-maps-qgis-plugin.md) to edit the data set. You may access your data set features by querying 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)

The data set ID is also needed to generate a tile set and state set. At any time in the development phase, you may use the Dataset List API to view details of the data sets you have generated. When you’re done using a data set, you can remove its resources using the Dataset Delete API.

### Tile sets

A tile set is a set of vector tiles that 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:

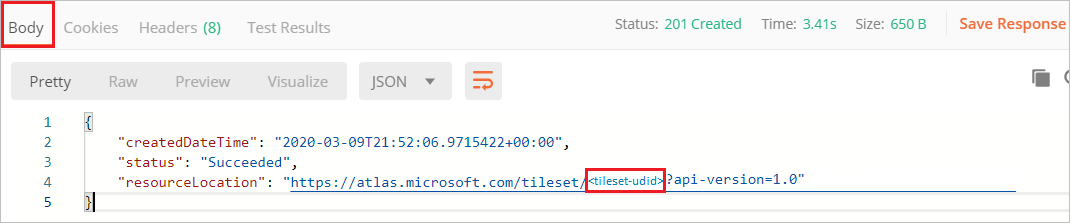
1. Open the Postman application. Select **New** again. In the **Create New** window, select **Request**. Enter a **Request name**, and select a collection. Click **Save**
2. Make a **POST** request in the builder tab. The request URL should look similar to the one below:

* 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 URL. Make a **GET** request at the URL, with your Azure Maps subscription appended to the end for authentication.

* https://atlas.microsoft.com/tileset/operations/<alphanumeric-value-status-URI>?api-version=1.0&subscription-key=<Azure-Maps-Primary-Subscription-key>

1. Upon a successful request, you’ll receive the tile set udid in the response body. For short, this udid is later referred to as the tile set ID.

* 
* data-management

Use the tile set ID to render tiles for indoor maps for the map data that was originally obtained from the DWG package. For example, you may render indoor maps using the Get Map Tile API. You may use the Tileset Delete API to delete tiles. If you delete an in use tile set, then those tiles won’t render on the map at the application runtime.

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.

[!Note] 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.

### State sets

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

The process of using the feature state set APIs is similar to using the data set and tile set APIs. You would first make a state set. And you would use the feature ID and the state set ID to update feature. The steps below explain how to create a state set, obtain the feature ID, update the feature ID state, and acquire the state of the feature.

1. Open the Postman application. Select **New**. In the **Create New** window, select **Request**. Enter a **Request name**, and select a collection. Click **Save**
2. If you don’t already have a feature ID, obtain a feature ID from a data set using the WFS API. First, 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 collections in your data set. Once you choose a collection, make a **GET** request to learn about the features inside the collections. The URL to use is below:

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

1. Choose a feature, and copy the **ID** of the feature. We’ll refer to the dynamic properties of this feature as states, and we’ll use the feature ID to modify its states.
2. 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.

* 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 stamp inside the JSON body 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. So, unlike the data set and the tile set APIs, there isn’t an API to fetch the different states of various features in your application. But, the Feature Get States API is sufficient for learning about the state of one feature. You can also delete the state set and its associated states using the Feature State Delete API.

### 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 limitations 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.

## Resource Administration

You can monitor the status of your map data using the list APIs available in the Data, Conversion, Dataset, Tileset, and Feature State services. You can review the relevance of the data. Then, you may update or delete the data, using the update and delete APIs, for the respective service. The Azure Maps QGIS plug-in is more convenient for administration purposes. The user interface lets you see real-time updates reflected based on changes to your data. Also, at this point, you would have become familiar with the Azure Maps APIs and the Azure Maps SDKs. And you may leverage more Azure Maps services. For example, if your application is “smart”, then you may incorporate IoT system powering location logic or produce map visualization.

## Next steps

Learn more about the different APIs mentioned in this application:

Conversion APIs

Dataset APIs

Tileset APIs

WFS APIs

Feature State APIs

To learn how to use the Azure Maps QGIS plug-in and how to style dynamic properties using the Private Atlas, see:

[Azure Maps QGIS Plug-in](azure-maps-qgis-plugin.md)

Implement dynamic styling for Private Atlas Indoor Maps

The Indoor Maps web SDK lets you use APIs in combination with the SDK to develop applications. To start learning more about this SDK, see:

Indoor Module SDK documentation