Implement dynamic styling for Private Atlas Indoor Maps | Microsoft Azure Maps

farah-alyasari

# Implement dynamic styling for Private Atlas Indoor Maps

Private Atlas lets you dynamically render certain parts of your indoor map data. For example, you may have indoor map data for a building with sensors collecting temperature information. You can render meeting rooms with styles based on the room temperature level. The Feature State API supports such scenarios in which the tileset features render according to their states. In this article, we’ll discuss how to dynamically render indoor map features based on associated feature states using the Feature State API and the Indoor Module module.

When a web application uses the Get Map Tile API to render indoor maps, you can further leverage the Feature State set service for dynamic styling. In particular, the Get State Tile API allows for control over the tileset style at the level of individual feature. So, the map rendering engine will not reparse the underlying geometry and data. This offers a significant boost in performance, especially in scenarios involving live data visualization.

## Prerequisites

To calls Azure Maps APIs, [make an Azure Maps account](quick-demo-map-app.md#create-an-account-with-azure-maps) and [obtain a primary subscription key](quick-demo-map-app.md#get-the-primary-key-for-your-account). This key may also be referred to as the primary key or the subscription key.

Enable Private Atlas, and use it to create an indoor map. The necessary steps are described in [make a Private Atlas account](how-to-manage-private-atlas.md) and [create an indoor map using Private Atlas](tutorial-private-atlas-indoor-maps.md). When you complete these steps, note your tile set identifier and feature state set identifier.

Build an application using the Indoor Maps module. The application doesn’t have to be complex, you can use the example in [how to use the indoor module](how-to-use-indoor-module.md#use-the-indoor-maps-module).

In this article, we use the Postman application to make API calls, but you can use any API development environment.

## Implement dynamic styling

Once you complete the prerequisites, you should have a simple web application, your tile set identifier, and state set identifier. The first steps below show you the key lines you should check in your code.

After the initial checks, we’ll exemplify an indoor map with meeting rooms. We’ll render the rooms dynamically based on their occupancy status. The meeting rooms with the “occupied” state set to “true” will be rendered as red. Those with a “false” state will be rendered as green.

### Perform initial checks

1. Check that your Indoor Manager contains your tile set ID and state set ID. If your indoor map has multiple levels, initialize the level control so you can see the different levels in your facility.

* const tilesetId = "<Your tilesetId>";  
  const statesetId = "<Your statesetId>"   
    
  const indoorManager = new atlas.indoor.IndoorManager(map, {  
   levelControl,  
   tilesetId,  
   statesetId  
  });

1. Check that your map instance is centered on the location of your indoor map, by setting the center parameter correctly. If you want, you may also use the bounds parameter to constrain your entire map data into a rectangular shape. You can use the following request to the Tileset List API to obtain the bounds for all your tileset IDs. Make sure you use the same bounds for the tile set ID that you’re providing in your code.

* GET https://atlas.microsoft.com/tileset?subscription-key=<Azure-Maps-primary-subscription-key>&api-version=1.0
* const subscriptionKey = "<your Azure Maps Primary Subscription Key>";  
    
  const map = new atlas.Map("map-id", {  
   //Use your map's coordinates:  
   //center: [ longitude, latitude ],  
   //bounds: [ longitude1 , latitude1 , longitude2 , latitude2 ]  
   //pick a style:  
   //style: "blank",  
   //pick a zoom level or leave it at auto:  
   //zoom: auto,  
   subscriptionKey  
  });

1. Check that you have dynamic styling enabled

* styleManager.setDynamicStyling(true);

### Select features

To implement dynamic styling, view your map features and select the feature identifier for the feature you want to style. You’ll use the feature to make a state for your feature state set. You can learn about your map features in several ways, which are described here:

* Use the Azure Maps QGIS plug-in. It gives you the ability to select map features for your map layers, using a friendly interface. For more information, see [how to use the Azure Maps QGIS plug-in](azure-maps-qgis-plugin.md).
* Make HTTP calls to the Azure Maps WFS service. It gives you systematic access to data set features, and it supports efficient management of feature state sets. For more information, see common WFS use cases
* Write customized code to visually select features in your rendered map. For example, add code to get the feature ID for a point on the map. In this article, we’ll show you one way to write custom code to learn about your feature.

The following script adds the ability to click on a point on the map and get the feature ID for the clicked point. This will help you visually select a feature, gather its identifier, and test the result of changing occupancy status for meeting rooms. In your application, you can insert the code below your Indoor Manager code block. Run your application, and 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")  
  
 var result = features.reduce(function (ids, feature) {  
 if (feature.layer.id == "indoor\_unit\_conference") {  
 console.log(feature);  
 }  
 }, []);  
});

For this exercise, choose two feature ID for two units.

### Set occupancy status

Create a new feature state set containing the feature ID you want in your feature state set. Update the state of the feature state set, and then reload your map to see the changes rendered on the map.

1. In 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 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=<dataset-udid>&subscription-key=<Azure-Maps-Primary-Subscription-key>

1. In the **Headers** of the **POST** request, set Content-Type to application/json. In the **Body**, provide the styles that you want to dynamically update. For example, you may use the following configuration. When you’re done, click **Send**.

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

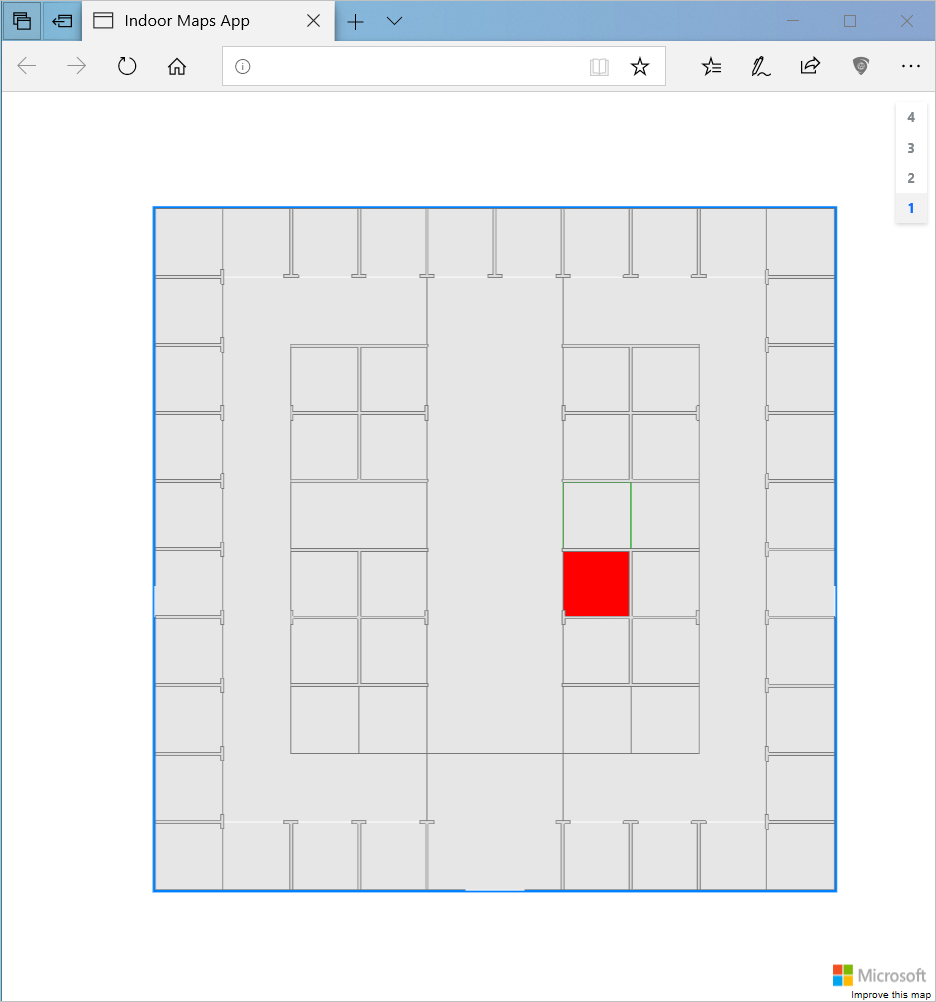
1. Copy the state set ID from the response body
2. Use the Feature Update States API to update the state. Pass the state set ID, data set ID, and feature ID for one of the two units. Append your Azure Maps subscription key. Here’s the URL of a **POST** request to update the state:

* https://atlas.microsoft.com/featureState/state?api-version=1.0&statesetID=<stateset-udid>&datasetID=<dataset-udid>&featureID=<feature-ID>&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 update will only be saved, if the time posted stamp is after the time stamp of the previous request. Pass the “occupied” keyname to update its value. Use the following JSON:

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

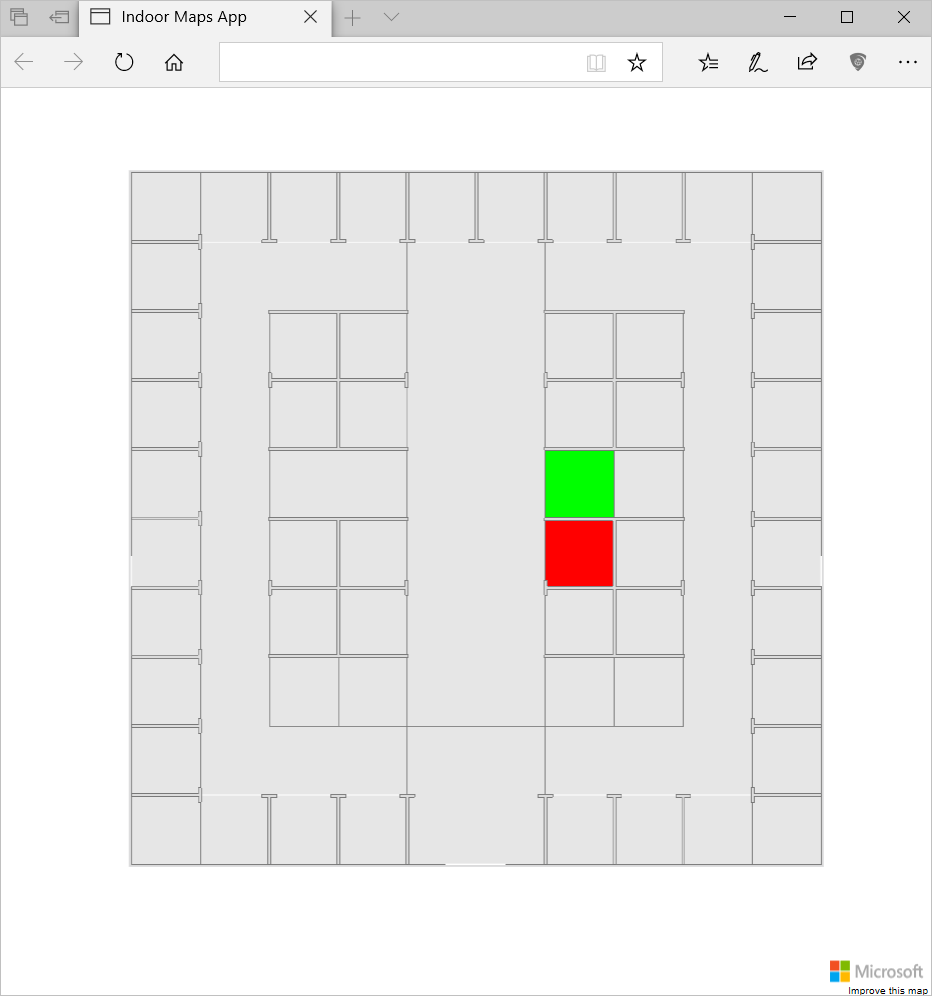
1. Refresh your HTML page, and you should see that the unit you selected is now rendered as red. Indicating that the occupancy status is busy, like in the image below:

* 
* busy room

1. Redo step 5 and 6 using the other feature ID, with the following JSON. Refresh your HTML page when you’re done.

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

1. You should see another room rendered in green, indicating that the room is available. Your map layout is probably different, but here’s how our layout looks like:

* 
* busy room

## Next steps

Learn more by reading:

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

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

See to the references for the APIs mentioned in this article:

[!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