VMSDK User Guide

Version 1.3 for Web

Contents

Welcome **4**

Audience 5

Supported browsers 5

Get started **5**

Install and run the Selection Reference App 5

Install and run the Wayfinding Reference App 6

Implement **6**

Overview 6

Setup 7

Create your HTML file 7

Mapbox access token 8

Create your JavaScript file 8

Load the VMD file and display your map 8

Legacy Support 9

Responding to Map View events and callbacks 9

Adding your own annotations to the map 10

Customizing your map's look and feel 10

Responding to errors that occur in the VMSDK 11

VMD parsing errors 11

Map display errors 11

Enable wayfinding **11**

Overview 11

Setup 12

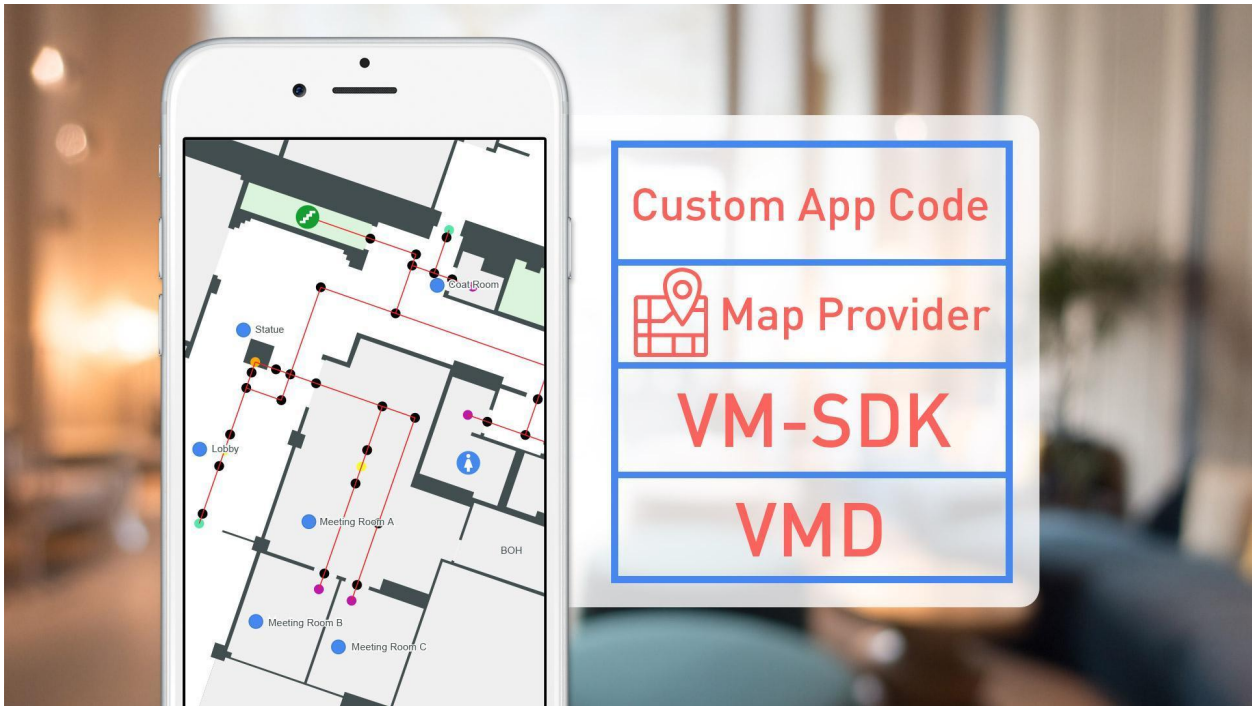
Client-Side 12

Server-Side 12

Finding waypaths 13

Client-Side 13

Find path 13
Get directions 13
Highlight path 14
Clear path 14
Server-Side 14
Find closest waypoint 14
Find path and directions 15
Override auto-generated wayfinding directions and landmark names 15
More Information 16
Appendix: Vector Map Tile Style Spec 17
Supported spec versions: 1.0 17
Style spec properties 17
Possible Style Layer ID Patterns 19



Welcome

Aegir uses groundbreaking technology to map venue spaces and give your users more choice, convenience, and control. Aegir's **Venue Map Data (VMD)** specification is the foundation, acting as a high-fidelity map and data repository.

VMD is a specification based upon industry standard digital mapping technology that provides a comprehensive geolocated data set for venue maps in which all geometries and points that make up the map shapes and points of interest are accessible, as well as metadata for use with the SDK to support extended applications. Potential use cases might include:

- Wayfinding
- Interior positioning, geofencing applications
- Room/space/unit selection
- Facilities management: environmental controls, housekeeping status and assignments, digital key hardware management, etc.

Aegir's **Venue Map Software Development Kit (VMSDK)** provides functionality for web development based on the VMD specification, including:

- Auto-generated wayfinding paths
- Auto-generated wayfinding directions with support for override from a separate data file
- Vector and raster map tiling on top of Mapbox Maps

Audience

VMSDK documentation is designed for people familiar with basic development for JavaScript and web applications.

Supported browsers

VMSDK supports the latest, stable releases of all major browsers. Consult your map provider's documentation for their own supported browser platforms.

Get started

The Venue Maps SDK for web comes with two sample reference apps:

- A simple "selection-reference-app" that includes a static HTML/JavaScript implementation of the Venue Maps SDK demonstrating how to do the following:
  - Show a map using a specific map provider, and display your custom map tiles
  - Load a VMD file
  - Apply custom styling to your map tiles
  - Handle errors that may occur
- A more complex "wayfinding-reference-app" that includes a Grails application implementation of the Venue Maps SDK which demonstrates how to do the following in addition to the above:
  - Load wayfinding data
  - Optionally load additional custom map data for directions and naming
  - Apply custom styling for your wayfinding

Install and run the Selection Reference App

1. Extract the web VMSDK zip file**.**
2. Generate a Mapbox access token. (The selection reference app uses Mapbox as an example map provider, so you will need to create an account and an access token with Mapbox.)
3. Edit the file **selection-reference-app/js/demo.js** and update the *accessToken* variable with the Mapbox access token that you created in Step 2.
4. Open the index.html file in a web browser.

NOTE: The selection reference app provides basic samples of how one might use the SDK components. It is not intended to be used in a production setting, nor should it be considered as a template for production code.

Install and run the Wayfinding Reference App

1. Download and install a Java SDK (version 1.8.0)
    a. The reference app backend is built with Grails 2.5.6, which requires the Java 8 SDK. However, this is not a requirement of the core VMSDK.
2. Extract the web VMSDK zip file**.**
3. Generate a Mapbox access token and generate Google Maps API Key. (The wayfinding reference app uses Mapbox and Google Maps as example map providers, so you will need to create an account and an access token with Mapbox, and an account and an API Key with Google.)
4. Edit the files **wayfinding-reference-app/web-app/js/demo_wayfinding.js and reference-app/web-app/js/demo_nojquery.js** and update the *accessToken* variable with the Mapbox access token that you created in Step 3.
5. Edit the file **wayfinding-reference-app\grails-app\views\layouts\googleMaps.gsp** and replace the “<INSERT YOUR API KEY>” placeholder with your Google Maps API Key that you created in Step 3.
6. In a command prompt or shell, change to the **wayfinding-reference-app/** subdirectory and run the reference app by executing this command: **grailsw run-app** for *nix systems, or **grailsw.bat run-app** for Windows. *Note: *nix users will have to modify the file access permission to allow execution using **chmod u+x**.*
7. After the app builds and starts, browse to http://localhost:8080/web-reference-app/ to interact with the reference app.

NOTE: The wayfinding reference app provides basic samples of how one might use the SDK components. It is not intended to be used in a production setting, nor should it be considered as a template for production code.

Implement

Overview

The SDK supports displaying and interacting with the map though a client-side JavaScript library. Documentation for the JavaScript is included in the SDK and can be found in the **/docs/javascript** subdirectory. You’ll need to start by creating HTML and JavaScript files, as well as deploying your VMD static assets on a web server.

Setup

Create your HTML file

VMSDK JavaScript calls require the use of certain JavaScript libraries. Specifically, this requires Mapbox GL JS (v0.44.2), Mapbox GL browser support (v1.3.0), and Aegir JavaScript files. Include the JavaScript files in the `<head>` of your HTML file.

Mapbox requires the Mapbox GL CSS (v0.44.2) stylesheet. Include the stylesheet `<link>` in the `<head>` of your HTML file.

In your HTML file you will need to add a DIV element with an ID of **map**. Your file should now look something like the following:

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Mapbox CSS -->
    <link href='https://api.tiles.mapbox.com/mapbox-gl-js/v0.44.2/mapbox-gl.css'
rel='stylesheet' />

    <!-- Mapbox JS -->
    <script
src='https://api.mapbox.com/mapbox-gl-js/plugins/mapbox-gl-supported/v1.3.0/mapbox-gl-s
upported.js'></script>
<script src='https://api.tiles.mapbox.com/mapbox-gl-js/v0.44.2/mapbox-gl.js'></script>
    <!-- VMSDK JS -->
    <script src='maps-sdk.js'></script>
    <title>Venue Map</title>
  </head>
  <body>
    <div id='map'></div>
  </body>
</html>
```

VMD files are deployed as .zip files. Depending on how your assets will be hosted, you can opt to unzip the files and load them individually through the SDK, or point the SDK at your .zip file and load everything automatically. In order to enable loading from the .zip file, there is an

additional JavaScript library called Zip JS that you must include in the `<head>` of your HTML file.

```
<!-- Zip JS -->
<script src="js/zip/zip.js"></script>
<script src="js/zip/zip-ext.js" ></script>
<script>zip.workerScriptsPath = "js/zip/";</script>
```

Mapbox access token

To use any of Mapbox's tools, APIs, or SDKs, you'll need a Mapbox access token. Mapbox uses access tokens to associate requests to API resources with your account. You can find all your access tokens, create new ones, or delete existing ones on your access tokens page.

Create your JavaScript file

Create your application's JavaScript file and add it to your HTML's head section. You can then begin utilizing the VMSDK library.

Load the VMD file and display your map

Now you're ready to write JavaScript to load your venue map data into your page. There are several implementation examples in the SDK's sample code:
reference-app\web-app\js\demo_*.js

```
//Specify where the VMD files are located: Consult the javascript docs for
'vmdFileCollection' to see the full list of accepted properties.
var collection = {
    zipURL: "https://server.com/my_vmd.zip"
};

//Specify other load configuration details: Consult the javascript docs for
'mapDataLoadConfig' to see the full list of accepted properties.

var loadConfig = {
    venueId: "MY_VENUE_ID",
    style: venueStyleURL //see section below for Customizing your map's look and feel
};
//Downloads and parses the VMD file(s), and has a callback for completion
aegir.loadVenueMapData(collection, loadConfig, function( vmd, errors){
    demo.didFinishLoadingVenueMapData(vmd, error);
});
```

Once the **aegir.loadVenueMapData** call has completed with no errors, call the **aegir.loadMapView** function to display the map on the page. The **aegir.loadMapView** function has several required parameters that are used to configure your map for display.

```
//Consult documentation for 'mapViewLoadConfig' for the full list of accepted
properties.
var mapViewConfig = {
   accessToken : accessToken,
   streetMapJSON: mapboxStyleURL,
   mapGlyphsURL : fontGlyphsURL,
   venueBaseURL: baseURL
};

//Consult documentation for 'mapViewCameraConfig' for the full list of accepted
properties.
var cameraConfig = {
   center: vmd.center,
   minZoom: 14,
   maxZoom: 21,
   pitch: 0,
   zoomLevel:18
};

//Loads the venue map data into a map view using the specified configuration and camera
location
aegir.loadMapView(mapViewConfig, cameraConfig, function(response, errors) {
    demo.didFinishLoadingMapView( response, errors );
 });
```

At this point, you should have a map display of the world with your map tiles superimposed.

Legacy Support

If you need to add support for parsing legacy venue map data files to your application, include the '**maps-legacy-sdk.js**' in your HTML file. The SDK will automatically detect if it's loading a zip file that contains legacy venue map data or new venue map data and process appropriately.

Note: Vector map tiles are not supported for legacy venue maps.

Responding to Map View events and callbacks

Map View events give your application code hooks for adding extra functionality when the user is interacting with the map. Map View events that may fire are as follows:

Event name	Description
`didSelectUnit`	Fired when the user selects some point of interest on

	the map -- e.g. a room
`didSelectAnnotation`	Fired when the user selects one of your map annotations.

Adding your own annotations to the map

You can programmatically add annotations to the map to suit your needs by creating a new `vmPointAnnotation` object and adding it to the map:

```
var element = document.createElement("div");
element.style.width = "50px";
element.style.height = "50px";
...
var annotation = {
   id: unit.id,
   htmlElement: element,
   location: unit.hotspotLocation || unit.labelLocation,
   floorId: unit.floorId
}

aegir.addAnnotation(annotation);
```

The look and feel of these map annotations can be styled just as any html element. When the user selects one of these annotations, a `didSelectAnnotation` event, described above in "**Responding to Map View events and callbacks**," will be fired.

Customizing your map's look and feel

If you use vector map tiles, you have great flexibility in styling your map. This customization also applies to raster map tiles, although it's much more limited. For more information, see **Appendix: Vector map tile style spec** below.

The easiest way to style your map is to create a Map Style JSON configuration file that follows the style spec in the appendix. For a full example, see **style_default.json** in the demo project. This is passed in through the aegir.loadVenueMapData function as params.style.

```
//load your custom style from style_default.json configuration file
var config = {
```

```
    style: document.location.origin + "/style_default.json",
    ...
}

aegir.loadVenueMapData(..., config, function(vmd, errors) {
    //load your custom style from style_default.json configuration file
});
```

Responding to errors that occur in the VMSDK

VMD parsing errors

If any errors are encountered while parsing your venue map data files during a call to **aegir.loadVenueMapData** in your client-side JavaScript, the callback function will return a list of errors with more detailed information.

Map display errors

If any errors are encountered when your map is loaded and rendered on-screen during a call to **aegir.loadMapView** in your client-side JavaScript, the callback function will return a list of errors with more detailed information.

Enable wayfinding

Overview

The VMSDK also supports more advanced features such as wayfinding. This is done through a client-side JavaScript library, and a server-side Java library. In order for the client-side JavaScript calls to function, the server needs to service the calls and return information to the client for a given map.

The map is loaded with JavaScript calls. During the loading process, the JavaScript makes a call to the server requesting that the web application load the map for the specified venue. Once the map is loaded, the user interacts with the map in the browser. When the user requests a wayfinding path between two locations, the client sends over a number of requests. The web application services the client calls for wayfinding and returns wayfinding data for the given requests.

In the VMSDK, the **libVMMS.jar** contains the classes used to process these calls and calculate the various wayfinding items. To use this JAR, you will need to include it in your web application for use in your application server.

The reference app provides a basic implementation for the purposes of demonstrating how these features can be used. Javadoc for the classes in **libVMMS.jar** is included in the VMSDK and can be found in the **/docs/javadoc** subdirectory.

Setup

Now you're ready to write code to load your waypoint data and map from the VMD file. Consult the example Grails application in the VMSDK. It can be found in the **/reference-app** directory.

Client-Side

Include the '**maps-wayfinding-sdk.js**' in your HTML file to enable wayfinding. Also, there are additional required properties that you need to set before calling **aegir.loadVenueMapData()**:

```
apiBaseURL          : "https://server.com/grails_application/",
assetBaseURL        : "https://server.com/wayfinding_images/",
```

The **apiBaseURL** is the endpoint where your API can be found. In the VMSDK's sample application, this is the path to the controller that handles the requests: **https://localhost:8080/web-reference-app/mapDemo**

The **assetBaseURL** is the endpoint where your start/end destination images can be found for wayfinding. This URL does not have to be part of your grails application, but can be URL(s) to static images instead.

Server-Side

The first request made by the client is for the server to preload the data required for wayfinding on the server side. The web application uses the parameters provided in the call to create a VMDFileCollection. This collection object is then used to load the map data in the web application.

Example:

```
VMDFile vmdFile = new VMDUrlFile(url);
VMDFile geoJSONFile = new VMDUrlFile(geoUrl);
VMDFile customMapInfoFile = new VMDUrlFile(customerMapInfoUrl);
VMDFileCollection collection = new VMDFileCollection(vmdFile, geoJSONFile);
mapInstance = com.aegir.vmms.vmd.model.Map.load(collection);
customMapInfoInstance = CustomMapInfo.load(customMapInfoFile, mapInstance, new
WaypointLabelOptions());
```

Finding waypaths

Client-Side

Find path

Call the **aegir.wf.findPath** function to find the waypath between two points. This requires **aegir.loadVenueMapData** and **aegir.loadMapView** to have completed. The start and end parameters will be the ID of the waypoint nodes.

```
aegir.wf.findWayPath(start, end, options)
```

Parameters:

```
start(string)= ...//ID of starting waypoint.
end(string)= ...//ID of ending waypoint.
options(Object)= ...//Parameters for aegir.wf.findPath function
options.callback(Function)= ...//Expecting callback() This is the
function that is called after data has been loaded.
```

Get directions

You can get a list of all direction data after **aegir.wf.findWayPath** has completed.

```
aegir.wf.getDirections()
```

Example:

```
$findPathCallback = function() {
   var directions = aegir.wf.getDirections();
}
```

Highlight path

You can highlight a segment of the wayfinding path, by using **aegir.wf.onSegmentSelected**. This requires the index of the instruction returned from **aegir.wf.getDirections**. For example: The first item returned would have an index of 0, the second item would be 1, etc. This also returns a callback containing an error array if necessary.

```
aegir.wf.onSegmentSelected(index, callback)
```

Example:

```
aegir.wf.onSegmentSelected(index, function(errorCollection) {
   // Handle errors.
});
```

Clear path

If you want to clear all wayfinding data and reset the map, you can use the **aegir.wf.reset** function.

```
aegir.wf.reset()
```

Server-Side

Find closest waypoint

The next request made by the client is for the server to return the closest waypoint. The web application passes the location and floorKey from the client to the mapInstance to find the closest waypoint.

Example:

```
log.debug("[findClosestWaypoint("+venueId+"]:
["+location.longitude+"."+location.latitude+"] "+floorKey);
```

```
def waypoint = mapInstance.findClosestWaypoint(location, floorKey);
```

Find path and directions

The next request made by the client is to request the path and directions. The web application receives this as one request but processes it as two actions. In order to find the path between two points, pass the two locations to the mapInstance using the findWaypathBetweenWaypoints method.

Example:

```
log.debug("[findPath("+venueId+", "+destinationStart+", "+destinationEnd+")]: ");
        mapInstance.labelWaypoints(new WaypointLabelOptions());

return mapInstance.findWaypathBetweenWaypoints(
mapInstance.findWaypointWithId(destinationStart),
mapInstance.findWaypointWithId(destinationEnd), new WayfindingOptions (true,false));
```

The web application then creates the directions for the path.

Example:

```
return mapInstance.createTurnByTurnDirectionsForWaypath(path, customMapInfoInstance);
```

Override auto-generated wayfinding directions and landmark names

You can use map information from a .json file to override the VMD's auto-generated wayfinding directions and landmark names. **Consult the example in the VMSDK demo**.

The data contained in your map info override file must be in JSON format, according to the following specs:

```
{
  "points": [
    {
      "id": "node_waypoint_b1_f1_517",
      "public-description": "the edge of the Basketball Court"
    },
    {
      "id": "<the ID of the waypoint>",
      "public-description": "<the description you want for this
landmark/waypoint">
    }
  ],
  "paths": [
    {
      "pathID": "node_path_b1_f1_722",
      "p1": "node_waypoint_b1_f1_473",
      "p2": "node_waypoint_b1_f1_567",
      "description-d1": "along the sidewalk",
      "description-d2": "along the sidewalk the other direction"
    },
    {
      "pathID": "<the ID of the path>",
      "p1": "<the ID of one of the waypoints>",
      "p2": "<the ID of the other waypoint>",
      "description-d1": "<description for traversing from P1 to P2>,
leave blank to auto-generate",
      "description-d2": "<description for traversing from P2 to P1>,
leave blank to auto-generate"
    }
  ]
}
```

More Information

For assistance with the Aegir VMSDK, related questions, or information about other Aegir products and services, visit https://support.aegirmaps.com/, contact Aegir Support at support@aegirmaps.com or call us at (901) 591-1631 between 9:00 am and 5:00 pm CST, Monday through Friday.

Appendix: Vector Map Tile Style Spec

Supported spec versions: 1.0

Style spec properties

Property-name	Required	Supported-layer-type	Description
id	yes		Identifier for this style definition. For venues with multiple styles, this identifier should be unique
name	no		Common name used to describe this style
version	yes		Spec format version. Certain versions of the VMSDK may only support certain versions of this spec format.
styles	yes		Container for list of style layer customizations
styles[].layer-id	yes	All	The ID of the style layer. See **Possible Style Layer ID Patterns** section below for acceptable values.
styles[].hidden	no	All	Specify as true to hide this layer. Default: false.
styles[].fill-color	no	Polygon	A HEX color to fill the polygon with. Default: NULL.
styles[].fill-pattern	no	Polygon	This is the name of an image from the style's sprite sheet to pattern fill the polygon with. Default: NULL.
styles[].line-color	no	Line	A HEX color to draw the line with. Default: NULL.
styles[].icon-name	no	Icon	This is the name of an image from the style's sprite sheet. Default: NULL.
styles[].font-name	no	Label	This is the name of a font to use for the labels. Default: NULL.
styles[].font-size	no	Label	The point size of the font. Default: NULL.
styles[].font-color	no	Label	A HEX color of the font: Default: NULL.

Property-name	**Required**	**Supported-layer-type**	**Description**
styles[].font-stroke-color	no	Label	A HEX color for the font outline. Default: NULL.
styles[].font-stroke-width	no	Label	This is the width of an outline for the font. Default: NULL.
styles[].max-text-width	no	Label	Controls automatic text wrapping within a label. Default: NULL.
wayfinding	no		Container for list of wayfinding style customizations.
wayfinding.path-stroke-width	no		A decimal value indicating how thick the stroke is for the default wayfinding path. Default: NULL.
wayfinding.path-stroke-color	no		A HEX color to draw the wayfinding path. Default: NULL.
wayfinding.path-stroke-alpha	no		A decimal value from 0 to 1 to indicate how transparent the default wayfinding path is. Default: NULL.
wayfinding.highlighted-path-stroke-width	no		A decimal value indicating how thick the stroke is for the highlighted section of the wayfinding path. Default: NULL.
wayfinding.highlighted-path-stroke-color	no		A HEX color to draw a highlighted section of the wayfinding path. Default: NULL.
wayfinding.highlighted-path-stroke-alpha	no		A decimal value from 0 to 1 to indicate how transparent the highlighted section of the wayfinding path is. Default: NULL.

Possible Style Layer ID Patterns

This is a list of possible layer-IDs that can be styles per the style spec above. Some layers apply only to raster or vector, while others apply to both. This is indicated in the 'tile-type' column below. This list is ordered by the zIndex they would appear in the map: layers with a higher Order will appear on top of those with lower values.

Supported wildcards:

1. FLOOR - the ID of the floor layer from the VMD (e.g. floor_b1_1)
2. BUILDING - the ID of the building layer from the VMD (e.g. building_1)

When wildcards are used, the specific style will be applied to all layers that match. For example, floor_elevators_[FLOOR] will be used for the floor_elevators_* layer on ALL floors in ALL buildings.

If you want to confine a unique style to a layer on a single floor, then don't use the wildcard. For example floor_elevators_floor_b1_2 would apply to the floor_elevators layer ONLY on floor 2 in building 1.

Order	Layer-ID	Tile-type	Layer-type	Description
1	background	All	Polygon	Background color of the entire map that is visible when the base map is hidden.
2	venue	vector	Polygon	“venue_outdoors” polygon.
3	outdoors	All	n/a	Raster map tiles that are part of the venue outdoor floor.
4	building_outlines_[BUILDING]	vector	Polygon	Building-outlines polygon for the given [BUILDING].
5	floor_outlines_[FLOOR]	vector	Polygon	Floor-outlines polygon for the given [FLOOR].
6	floor_elevators_[FLOOR]	vector	Polygon	Elevator polygons for the given [FLOOR].
7	floor_stairwells_[FLOOR]	vector	Polygon	Stairwell polygons for the given [FLOOR].
8	floor_restrooms_[FLOOR]	vector	Polygon	“Restroommen” and “restroomwomen” polygons for the given [FLOOR].
9	floor_walkways_[FLOOR]	vector	Polygon	Walkway polygons for the given [FLOOR].

Order	Layer-ID	Tile-type	Layer-type	Description
10	floor_fixtures_[FLOOR]	vector	Polygon	Floor fixture polygons for the given [FLOOR].
11	floor_non_public_units_[FLOOR]	vector	Polygon	Non-public unit polygons for the given [FLOOR].
12	floor_open_to_below_units_[FLOOR]	vector	Polygon	“Open to below” unit polygons (such as open atrium spaces) for the given [FLOOR].
13	floor_other_rooms_[FLOOR]	vector	Polygon	“Other room” polygons for the given [FLOOR].
14	floor_rooms_[FLOOR]	vector	Polygon	Room polygons for the given [FLOOR].
15	floor_water_[FLOOR]	vector	Polygon	Fixtures where category=Water for the given [FLOOR].
16	floor_openings_[FLOOR]	vector	Line	Floor openings for the given [FLOOR].
17	floor_amenities_[FLOOR]	vector	**	Floor amenities for the given [FLOOR].
18	floor_selected_unit_[FLOOR]	vector	Polygon	This is for the style of the actively selected polygon used during wayfinding & room selection for the given [FLOOR].
19	floor_shadows_[FLOOR]	vector	n/a	Raster map tiles that are overlaid on top of existing vector data for the given [FLOOR]
20	floor_labels_[FLOOR]	All	Label	Labels for the given [FLOOR].
21	floor_icons_[FLOOR]	All	Icon	Icons for the given [FLOOR].