aegir

# VMSDK User Guide

Version 1.1.1 for Web

## Contents

**Welcome** 3

Audience 4

Supported browsers 4

**Get started** 4

**Implement** 5

Overview 5

Setup 5

Create your HTML file 5

Mapbox access token 7

Create your JavaScript file 7

Load the VMD file and display your map 7

Client-Side 7

Server-Side 9

Enable Wayfinding 10

Client-Side 10

Find Path 11

Get Directions 12

Highlight Path 12

Clear Path 13

Server-Side 13

Find Closest Waypoint 13

Find Path and Directions 13

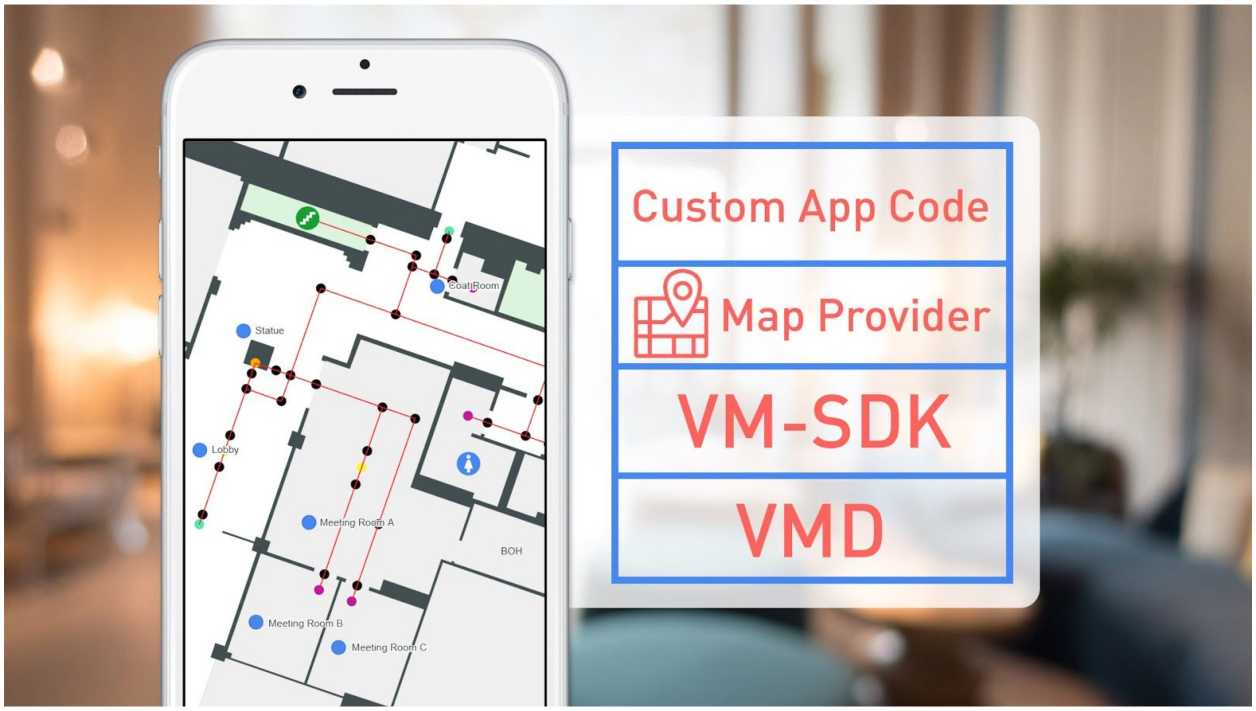
Find Heading Between Points 14

Override auto-generated wayfinding directions and landmark names 15

Customizing your map's look and feel 16

Responding to Errors that occur in the SDK 16

VMD parsing errors 16
Map display errors 16
**More Information** **17**
**Appendix: Vector Map Tile Style Spec** **18**
Supported spec versions: 1.0* 18
Style spec properties 18
Possible Style Layer ID Patterns 20



## 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 applications supported include:

- Wayfinding
- Room/space/unit selection

Aegir's **Venue Map Software Development Kit (VMSDK)** provides a suite of 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
- Support for SVG tiling on top of Open Street 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 platforms.

## Get started

With the Venue Maps SDK Reference App, you'll see how to:

- Show a map using a specific map provider, and display your custom map tiles
- Load the VMD file with wayfinding data
- Optionally load additional custom map data for directions and naming
- Allow custom styling for your map tiles
- Allow custom styling for your wayfinding
- Handle errors that occur

First, install and run the reference app:

1. Download and install a java SDK (version 1.7.0)

   a. The reference app backend is built with Grails 2.2.1, which requires the Java 7 SDK. However, this is not a requirement of the core VMSDK.

2. Extract the web VMSDK zip file**.**
3. The reference app uses Mapbox so you will need to generate a Mapbox Access token
4. Once you have a Mapbox access token, edit the **reference-app/web-app/js/demo_aegirh.js** file and update the *accessToken* variable.
5. In a command prompt or shell, change to the **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.
6. After the app builds and starts, browse to http://localhost:8080/web-reference-app/ to interact with the reference app.

The 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 as well as performing wayfinding 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.

In the SDK, 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 the these features can be used. Javadoc for the classes in **libVMMS.jar** is included in the SDK and can be found in the **/docs/javadoc** subdirectory.

## Setup

### Create your HTML file

Now you’re ready to write code to load your waypoint data and map from the VMD file. Consult the example in the SDK demo: **http://localhost:8080/web-reference-app.**

VMSDK JavaScript calls require the use of certain JavaScript libraries. Specifically, this requires jQuery, 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' />

    <!-- jQuery JS -->
    <script src='https://code.jquery.com/jquery-3.3.1.min.js'></script>

    <!-- 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>

    <!-- Aegir JS -->
    <script src='maps-sdk.js'></script>
    <script src='maps-wayfinding-sdk.js'></script>

    <title>Venue Map</title>
  </head>
  <body>
    <div id='map'></div>
  </body>
</html>
```

### 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

Begin by defining each of the required **global** variables. **Consult the example in the SDK demo**.

```
venueId(string)= ...//Id of the venue you want to appear on the map

getHeadingURL(string)= ...//Path to controller method that calls
headingBetweenPoints. Example:
"/web-reference-app/mapDemo/findHeadingBetweenPoints"

postDataURL(string)= ...//Path to controller method that calls
loadMapInstance. Requires: venueId
Example: "/web-reference-app/mapDemo/loadMapInstance"

customMapInfoURL(string)= ...//Path to customMapInfo.json
Example: "/web-reference-app/aegir_maps/customMapInfo.json"
```

## Load the VMD file and display your map

This section provides an overview of how the map display implementation functions in the reference app, as well as how to add these capabilities to your own web application.

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 VMD for the specified venue. Once the map is loaded, the user interacts with the map in the browser

### Client-Side

Once all required global variables are set, call the **aegir.loadData** function to load and parse required VMD files. The **aegir.loadData** call requires the **postDataURL** global variable to be defined and the server-side request described in the next section. This also has options that are used to customize map display. See below for a list and description of all required parameters.

```
aegir.loadData(xml, geoJSON, params, callbackFunction)
```

Parameters:

```
xml(string)= ...//Path to XML file
Example: "/web-reference-app/aegir_maps/AEGIRH/venue_map_AEGIRH.xml"

geoJSON(string)= ...//Path to geojson file.
Example:
"/web-reference-app/aegir_maps/AEGIRH/venue_map_AEGIRH.geojson"

params(Object)= ...//Required params for aegir.loadData function.

params.style(string)= //Path to layer style file.

callback(Function)= ...//Expecting callback(data, error) Function that
returns a data object and an error array.
```

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

```
aegir.loadMap(params, center_lat, center_long, callbackFunction)
```

Parameters:

```
params(Object)= ...//Property returned from loadData callback. This is
the same as aegir.loadData.params.


//Mapbox setup

params.accessToken(string)= //Mapbox access token.

params.defaultMap(boolean)= //Set to true for OpenStreetMap to load. Set
to false for OpenStreetMap to not load.

params.mapGlyphsURL(string)= //URL of label font to be displayed.
Example: "mapbox://fonts/mapbox/{fontstack}/{range}.pbf"

params.mapSpritesURL(string)= //URL of map icons to be displayed.
Example: "/web-reference-app/aegir_maps/AEGIRH/icons/icons"
```

```
params.streetMapJSON(string)= //URL of OpenStreetMap style JSON to be
displayed.
Example: "/web-reference-app/aegir_maps/map_style_default.json"

params.streetMapURL(string)= //URL of OpenStreetMap to be displayed.
Example: "mapbox://mapbox.mapbox-streets-v7"


//Venue Map setup

params.tilesetURL(string)= //URL of tile set to be displayed. {FLOOR} is
required to display each floor tile.
Example: "/web-reference-app/aegir_maps/AEGIRH
/vector-tiles/{FLOOR}/{z}_{x}_{y}.mvt"

params.commonTilesetURL(string)= //URL of common tile set to be
displayed. {FLOOR} is required to display each floor tile.
Example: "/web-reference-app/aegir_maps/AEGIRH
/vector-common/{FLOOR}/{z}_{x}_{y}.mvt"

params.buildingOutlines(string)= //Path to building outlines.
Example:
"/web-reference-app/aegir_maps/AEGIRH/vector-tiles/building_outlines_1/
{z}_{x}_{y}.mvt"


//Illustrative/artistic map setup

params.rasterTilesetURL(string)= //(Optional) URL of the tileset for
artistic/raster tiles to be displayed. {FLOOR} is required to display
each floor tile.
Example: "/web-reference-app/aegir_maps/AEGIRH
/vector-tiles/{FLOOR}/{z}_{x}_{y}.png"

params.rasterBuildingOutlinesURL (string)= //(Optional) URL of the
tileset for artistic/raster tiles to be displayed for the building
outlines.
Example: "/web-reference-app/aegir_maps/AEGIRH
/vector-tiles/building_outlines_1/{z}_{x}_{y}.png"


//Initial floors that will be displayed

params.initialIndoorFloors(array)= //Array of indoor floor Ids to be
displayed on load.
Example: ["floor_b1_1"]

params.initialOutdoorFloors(array)= //Array of outdoor floor Ids to be
displayed on load.
Example: ["floor_vo_1"]


center_lat(string)= ...//Property returned from aegir.loadData callback.
Center latitude coordinate.
```

```
center_long(string)= ...//Property returned from aegir.loadData callback.
Center longitude coordinate.

callback(Function)= ...//Expecting callback(response, errors) Function
that returns a response and an error array.
```

### Server-Side

The first request made by the client is for the server to load the map instance. The web application uses the parameters provided in the call to create a VMDFileCollection.  This collection object is this used to load the map instance in the web application. The endpoint to this method must be the same as the global **postDataURL** in the client-side javascript.

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());
```

At this point, you should have a map that is showing the world with your map tiles superimposed on top.

## Enable Wayfinding

This section provides an overview of how the wayfinding implementation functions in the reference app, as well as how to add these capabilities to your own web application.

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.

### Client-Side

If you want to add wayfinding capabilities to your mapview, you can begin by adding the **maps-wayfinding-sdk.js** file to your HTML page. Then, there are some additional required global variables to define in your javascript. **Consult the example in the SDK demo**.

```
startPin(string)= ...//Path to marker image designated as the start
marker. Example: "/web-reference-app/assets/mapDemo/greenPin.png"

endPin(string)= ...//Path to marker image designated as the end marker.
Example: "/web-reference-app/assets/mapDemo/redPin.png"

defaultPin(string)= ...//Path to marker image designated as the default
marker. Example: "/web-reference-app/assets/mapDemo/defaultPin.png"

closestWaypointURL(string)= ...//Path to controller method that calls
findClosestWaypoint.
Example: "/web-reference-app/mapDemo/findClosestWaypoint"

customMapInfoURL(string)= ...//Path to customMapInfo.json
Example: "/web-reference-app/aegir_maps/customMapInfo.json"

findPathURL(string)= ...//Path to controller method that calls findPath.
Example: "/web-reference-app/mapDemo/findPath"
```

### Find Path

To set a start point, end point, or default marker, use the **aegir.setMarkerType** function. Set the marker type you would like to drop on the map, and the function to call after the item on the map has been selected.

```
aegir.setMarkerType(markerType(string), callback function(Function))
```

```
// Use one of the following for your marker types. default: null.

'start'   - Set marker type to the starting waypoint marker image.
'end'     - Set marker type to the destination waypoint marker image.
'default' - Set marker type to a default marker image.
```

Example:

```
aegir.wf.setMarkerType('start', $.loadSelection);

$.loadSelection = function (item) {
   // Add functionality for after map selection. Item will be the object that was
selected
   on the map.
}
```

Call the **aegir.wf.findPath** function to find the waypath between two points. This requires **aegir.loadData** and **aegir.loadMap** to have completed. The start and end parameter will be the id of the item returned to your callback function from setMarkerType.

```
aegir.wf.findPath(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 findPath 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.directionHighlight**. 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.directionHighlight(index, callback)
```

Example:

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

### Clear Path

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

```
aegir.clearPath()
```

## 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);
```

## Find Heading Between Points

An additional request made by the client is to request the heading between two points. In order to find the path between two points, pass the two locations to the MapUtil class using the headingBetweenPoints method.

Example:

```
double lat1 = Double.parseDouble(params.lat1)
double lon1 = Double.parseDouble(params.lon1)
double lat2 = Double.parseDouble(params.lat2)
double lon2 = Double.parseDouble(params.lon2)

LatLng destinationStart = new LatLng(lat1, lon1);
LatLng destinationEnd = new LatLng(lat2, lon2);

return MapUtil.headingBetweenPoints(destinationStart, destinationEnd);
```

### 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 SDK demo**. The path to this file must be defined in the **customMapInfoURL** defined in the required global variables section earlier in this document.

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"
    }
  ]
}
```

## Customizing your map's look and feel

If you use vector map tiles, as opposed to raster 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.loadData function as params.style.

```
//load your custom style from style_default.json configuration file
var loadOptions = {
   style: document.location.origin + "/style_default.json",
   defaultMap: true,
   callback: function() {

   }
}

aegir.loadData(xmlURL, geoJSONURL, loadOptions, function(mapData, errors) {
   //load your custom style from style_default.json configuration file
});
```

## Responding to Errors that occur in the SDK

### VMD parsing errors

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

## More Information

For assistance with the SDK or any related question, contact Aegir at support@aegirmaps.com or at (901) 591-1624 between 9:00 am and 4:30 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 SDK 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 layers 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]. |