VMSDK User Guide

Version 1.3 for iOS

Contents

Welcome **4**

Audience 5

Supported platforms 5

Get started **5**

Install and run the reference app 5

Implement **6**

Load the VMD file and display your map 6

Legacy Support 7

Customizing your map's look and feel 8

Global map styling 8

Styling individual map elements 8

Responding to VMMapView events and callbacks 9

Map loading complete 9

Changes in map position 9

Room selection/highlighting 9

Adding your own annotations to the mapview 10

Responding to errors that occur in the SDK 11

VMD parsing errors 11

Map display errors 11

Enable Wayfinding **12**

Handle Wayfinding Events 12

Override auto-generated wayfinding directions and landmark names 13

Responding to wayfinding events and callbacks 14

Changing floors for wayfinding 14

Responding to errors that occur in wayfinding 15

Wayfinding errors 15

Customizing wayfinding look and feel 15

Landmark customization 16

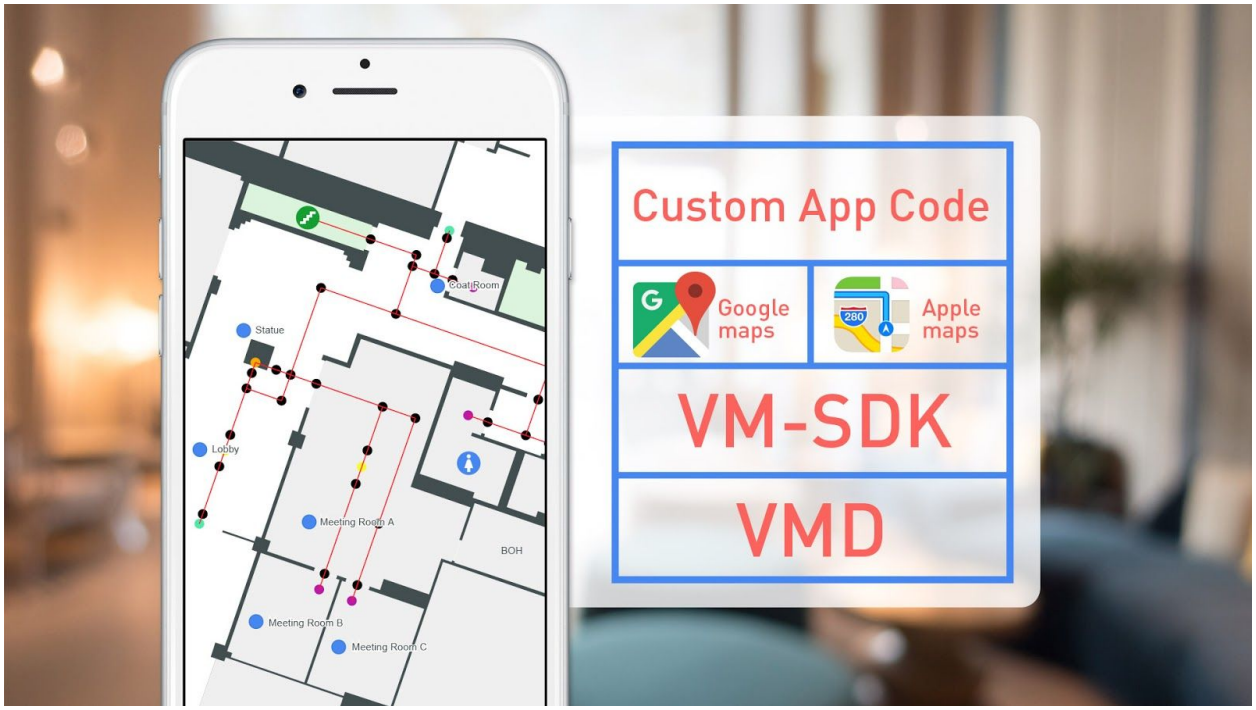
More Information 16

Appendix: Vector Map Tile Style Spec 17

Supported spec versions: 1.0, 1.1 17

Style spec properties 17

Possible Style Layer ID Patterns 21



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) for iOS** provides functionality for mobile app 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 Vector and Raster map tiling on top of Google and Apple Maps

Audience

VMSDK documentation is designed for people familiar with basic mobile development for iOS.

Supported platforms

VMSDK supports iOS 8.x+. 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

Install and run the reference app

1. Make sure you have version 10.2 or later of Xcode.
2. If you don't already have the CocoaPods tool, install it on macOS by running this command from the terminal:
   ```
   sudo gem install cocoapods
   ```
3. Extract the iOS VMSDK zip file.
4. In the extracted directory, navigate to **vmsdk-sampleapp**.
5. Install the dependencies using CocoaPods:
   ```
   pod install
   ```
6. In the extracted directory, navigate to **vmsdk-sampleapp/VMMS-Demo.xcworkspace** and launch it in Xcode.

7. The reference app uses the Google Maps SDK for iOS, so you will need to generate a Google API Key.
8. Once you've generated an API key, open **Supporting Files/VMMSDemoConstants.h** and update the GMS_API_KEY variable.
9. Build and run the app on the simulator.

Implement

Load the VMD file and display your map

Now you're ready to write code to load your waypoint data and map data into the VMD file. Consult the example in the SDK demo: **Controller/MapViewController.m.**

Implement the VMMSMapDelegate protocol to be notified when the map is finished loading.

You can also load custom map info from a file to override any auto-generated labeling information or to provide additional information where the SDK cannot determine useful points of interest for specific sections of your map.

```
//load the VMD from the zip file
NSString *zipFilePath = [[NSBundle mainBundle]
pathForResource:@"venue_map_sample" ofType:@"zip"];
id<VMDFile> localFile = [[VMDLocalZipFile alloc]
initWithAbsoluteFilePath:zipFilePath];

//create a file collection
VMDFileCollection *fileCollection = [[VMDFileCollection alloc]
initWithBaseZipFile:localFile];

[VMMSMap load:fileCollection delegate:self];

...
...
...

//implement VMMSMapDelegate  to get notified when the map is done loading
/**
 * Called when the VMD file is done loading SUCCESSFULLY
   @param map a VMMSMap object with waypoint data
 */
```

```
- (void)didFinishLoadingMap:(VMMSMap *)map
customMapInfo:(VMMSCustomMapInfo *)customMapInfo
{
     //create your VMMapView and get it all setup.
     //This will show floor 1, by default
    self.vmMapView = [[VMVectorMapView alloc]
initWithFrame:self.baseMap.frame vmd:self.vmd];

    //configure where your map tiles reside. This can be within your
application bundle, or a remote URL.
    self.vmMapView.tileBaseURL = [NSBundle.mainBundle.bundleURL
URLByAppendingPathComponent:@"vector_tiles"].absoluteString;

    //configure map label & icon data url, again either within your
application bundle, or a remote URL
    self.vmMapView.vectorCommonBaseURL = [NSBundle.mainBundle.bundleURL
URLByAppendingPathComponent:@"vector_common"].absoluteString;

    //other config options (see class documentation for the full list)
    self.vmMapView.minZoom = 17.0;
    self.vmMapView.maxZoom = 23.0;
    self.vmMapView.delegate = self; //receive map view callbacks
    self.vmMapView.map = self.map;

    //finally, add your mapview to the screen
    [self.vmMapView attachInView:self.view aboveView:self.baseMap];

    //set your map's initial position
    [self.vmMapView setMapPositionWithTarget: self.map.centerPoint
bearing:self.map.initialRotation zoom:self.map.initialZoom];
}
```

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 '**VMSDK-Legacy.framework**' in your project. 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.

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.

Global map styling

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.

```
//load your custom style from style_default.json configuration file
VMVenueStyle* style = [[VMVenueStyle alloc] initWithConfig:[NSBundle.mainBundle
URLForResource:@"style_default" withExtension:@"json" ]
venueId:@"venue_map_sample"];

//apply the style to your map
VMMapView* mapView = ...
mapView.style = venueStyle;
```

Styling individual map elements

You can now add custom styles to individual map elements. This allows you to override the overall map style defined in your map's VMVenueStyle configuration for a single element.

```
//get the map element
VMMSMapUnit* element = ...
VMMapView* mapView = ...

//create your custom style
VMVenueLayerStyle* elementStyle = [VMVenueLayerStyle new];
elementStyle.fillColor = UIColor.redColor;
elementStyle.fontColor = UIColor.blueColor;
//... see class documentation for a full list of styleable attributes

//apply the style to the unit. If the map unit is not visible, it will be
applied the next time it is shown.
[mapView setStyle: elementStyle forUnit: element];
```

Responding to VMMapView events and callbacks

There are numerous ways to customize the behavior of your mapview by responding to the following events:

Map loading complete

In some scenarios, you may want to wait until the mapview has completed loading before proceeding to a next step. You can wait for the didFinishLoadingMap callback.

```
/// Called when map view has finished loading so you can do any additional
setup
///
/// - Parameter map: the map
@objc optional func didFinishLoadingMap(map: VMMapView);
```

Changes in map position

Anytime the map's position changes, you can act accordingly.

NOTE: If your VMMapView is overlaid on another provider's map (such as Google Maps or Apple Maps), this is a good place to ensure the map positions stay in sync with each other.

```
/// Called when the map position changes
///
/// - Parameters:
///   - newLocation: the new map location
///   - newZoom: the new map zoom
///   - newBearing: the new map bearing
///   - newTilt: the new map tilt
@objc optional func didChangeCameraPosition(
                            toLocation newLocation: CLLocationCoordinate2D,
                                    toZoom newZoom: Float,
                              toBearing newBearing: Double,
                                    toTilt newTilt: Double);
```

Room selection/highlighting

When the user taps the map on a specific point or room, you can respond to those events appropriately. If you return true to canSelectUnit, the map will also highlight the selected shape using the color defined in your Map Style json for the layer-id of "floor_selected_unit_[FLOOR]". See “Customizing your map's look and feel” above for more information.

```
/// Called to see if it’s possible to select a unit
///
/// - Parameter unit: the unit to select
/// - Returns: true to allow selection, false otherwise
@objc optional func canSelectUnit(_ unit: VMMSMapUnit ) -> Bool;

/// Called after a new unit has been selected
///
/// - Parameter unit: the unit that is selected
@objc optional func didSelectUnit(_ unit: VMMSMapUnit? );
```

Adding your own annotations to the mapview

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

```
let marker = VMPointAnnotation();

marker.coordinate = ...//set the location you want the annotation to appear on
the map

marker.title = //give it a title, which you can use to reference in additional
callbacks

marker.floorNumber = //give the icon a floor number that it should be
displayed on (it's going to be removed when you're not looking at that floor
of the map)

marker.floorId = //set the floor id to the VMMSBaseFloor object's uid that
this marker belongs on

let mapView : VMMapView? = ...

mapView?.addAnnotation(marker);
```

You can configure the appearance of your custom annotation using these callbacks:

```
/// Called to provide a custom image for a point annotation
///
/// - Parameter annotation: the annotation
/// - Returns: the custom image
@objc optional func imageForPointAnnotation ( _ annotation: VMPointAnnotation
) -> UIImage?;

/// Called to provide a custom view for a point annotation
///
/// - Parameter annotation: the annotation
/// - Returns: the custom view
@objc optional func viewForPointAnnotation( _ annotation: VMPointAnnotation )
-> UIView?;
```

Responding to errors that occur in the SDK

There are numerous instances where an error could occur within the VMSDK at any of the many steps above. You can be notified of the error by implementing any of the following callbacks:

VMD parsing errors

If any errors are encountered while parsing your venue map data files, this method will be called within the SDK with more detailed information about the error:

```
/**
 * Called when the VMD file FAILS to load
 * @param error the exception that was raised during load
 */
- (void)didFailToLoadMapWithError:(NSError*)error;
```

Map display errors

If any errors are encountered when your map is loaded and rendered on screen through the VMMapView object, this method will be called within the SDK with more detailed information about the error:

```
/// Called when the mapview fails to load
///
/// - Parameter error: the error that caused the failure
/// - Since: 1.2
```

```
@objc optional func didFailToLoadMap(error: Error);
```

Enable Wayfinding

You can add wayfinding capabilities to your mapview to enable your application to provide turn-by-turn paths and directions:

```
//Create a VMVectorWalkingPathOverlay object, which will handle interacting
with the map to select start/endpoint locations for wayfinding
self.walkingPathOverlay = [[VMVectorWalkingPathOverlay alloc]
initWithMap:self.vmMapView];
self.walkingPathOverlay.map = self.map;
self.walkingPathOverlay.delegate = self;
self.walkingPathOverlay.currentOutdoorFloor = initialOutdoorFloor;
 .. and so on
```

Handle Wayfinding Events

Implement the VMMSWayfindingDelegate protocol to get notified of any callbacks from the SDK for wayfinding:

```
VMMSMap* map = ...
...
...
...
/**
* Called when wayfinding path is found
* @param waypath the Waypath that leads from the starting point to the ending
point
*/
- (void)didFinishFindingWaypath:(VMMSWaypath *)waypath {
{
     //if you want to add turn by turn directions
    [self.map createTurnByTurnDirectionsForWaypath:waypath
                                withCustomMapInfo:nil
                                       andOptions:nil
                                         delegate:self];
}
/**
* Called when turn by turn directions are completed
* @param turnByTurnDirections list of directions
*/
```

```
- (void)didFinishCreatingTurnByTurnDirections:(NSArray<VMMSMapDirectionStep *>
*)directions
{
    //display path on the map now
    //tell your VMWalkingPathOverlay about the waypath  & directions
    [self.walkingPathOverlay setWaypath:waypath andDirections:directions];

    //Figure out the first floor in the waypath
    VMMSWaypathSegment* firstSegment = directions.firstObject.segment;
    VMMSMapBuildingFloor* vmdFloor = [self.map
                                findFloorWithId:firstSegment.floorId];

    //update the map and the path overlay to show the new floor
    self.walkingPathOverlay.currentFloor = vmdFloor;
    self.vmMapView.activeIndoorFloors = @[vmdFloor];

    //trigger the walking path to draw all of the waypath
    //data for the active floors
    [self.walkingPathOverlay togglePathForIndoorFloors:
                                 self.vmMapView.activeIndoorFloors
                          andOutdoorFloors:
                                 self.vmMapView.activeOutdoorFloors];

    //if you want to automatically focus on the first segment of the waypath
    [self.walkingPathOverlay onSegmentSelected:firstSegment];

}
```

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. See this full example: **Controller/MapViewController.m.**

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"
    }
  ]
}
```

Responding to wayfinding events and callbacks

Changing floors for wayfinding

When you have a waypath that spans multiple floors, the default behavior for the VMMapView is to draw a button on the map that looks like this:

You can provide your own image that matches your own branding. Additionally, when the user selects that button, you must implement that behavior as well by specifying what floor to change to, etc. See **BaseMapViewController.m** (iOS) for an example of how to appropriately respond to a floor change event.

```
/// Called to provide a custom image for the floor change annotation button
///
/// - Parameter annotation: the annotation
/// - Returns: the custom image
@objc optional func imageForFloorChangeAnnotation( _ annotation:
VMFloorChangePointAnnotation) -> UIImage?;


/// Called when floor change annotation is selected
///
/// - Parameter annotation: the annotation
@objc optional func didSelectFloorChangeAnnotation( _ annotation:
VMFloorChangePointAnnotation);
```

Responding to errors that occur in wayfinding

Wayfinding errors

Errors may occur during wayfinding, usually if no paths exist between your selected start & end destination. This method will be called within the SDK with more detailed information about the error:

```
/**
 Called when an error occurred while attempting to find a waypath.

 @param error The exception that was raised during waypath finding.
 */
- (void)didFailToFindWaypathWithError:(NSError *)error;

/**
 * Called when the system is unable to generate turn by turn directions
 * @param error the exception that was raised
 */
- (void)didFailToCreateTurnByTurnDirectionsWithError:(NSError*)error;
```

Customizing wayfinding look and feel

All styling for wayfinding is now done through new styling properties added in v1.2 to the “wayfinding” section of the Vector map tile spec. For more information, see **Appendix: Vector map tile style spec** below.

Landmark customization

In the turn-by-turn directions provided by the SDK for wayfinding, there are usually points of interest, or landmarks, that are part of each step and refer to actual places on the map. You can add special icons to the map to further highlight your landmarks:

```
/// Called to provide a custom image for a landmark annotation
///
/// - Parameter annotation: the annotation
/// - Returns: the custom image
@objc optional func imageForLandmarkAnnotation( _ annotation:
VMLandmarkAnnotation) -> UIImage?;
```

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, 1.1

As new styleable features are added, we will attempt to maintain backwards compatibility with older versions of this specification, however newer styling features may not be available in older versions.

Style spec properties

Property-name	Required	Supported layer-type	Spec version	Description
id	yes		1.0	Identifier for this style definition. For venues with multiple styles, this identifier should be unique
name	no		1.0	Common name used to describe this style
version	no		1.1	Spec format version. Certain versions of the SDK may only support certain versions of this spec format. Default: 1.0
styles	yes		1.0	Container for list of style layer customizations
styles[].layer-id	yes	All	1.0	The ID of the style layer. See Possible style layers section below for acceptable values.
styles[].hidden	no	All	1.0	Specify as true to hide this layer. Default: false.
styles[].fill-color	no	Polygon	1.0	A HEX color to fill the polygon with. Default: NULL.
styles[].fill-pattern	no	Polygon	1.0	This is the name of an image from the style's sprite sheet to pattern fill the polygon with. Default: NULL.
styles[].outline-color	no	Polygon	1.1	This is a HEX color to draw an outline around the polygon with. Default: NULL

styles[].line-color	no	Line	1.0	A HEX color to draw the line with. Default: NULL.
styles[].icon-name	no	Icon	1.0	This is the name of an image from the style's sprite sheet. Default: NULL.
styles[].font-name	no	Label	1.0	This is the name of a font to use for the labels. Default: NULL.
styles[].font-size	no	Label	1.0	The point size of the font. Default: NULL.
styles[].font-color	no	Label	1.0	A HEX color of the font: Default: NULL.
Property-name	**Required**	**Supported layer-type**	**Spec version**	**Description**
styles[].font-stroke-color	no	Label	1.0	A HEX color for the font outline. Default: NULL.
styles[].font-stroke-width	no	Label	1.0	This is the width of an outline for the font. Default: NULL.
styles[].max-text-width	no	Label	1.0	Controls automatic text wrapping within a label. Default: NULL.
wayfinding	no		1.0	Container for list of wayfinding style customizations.
wayfinding.path-stroke-width	no		Deprecated 1.1	A decimal value indicating how thick the stroke is for the default wayfinding path. Default: NULL.
wayfinding.path-stroke-min-width	no		1.1	This is a decimal value indicating how thick the stroke is for the default wayfinding path at the map's minimum zoom level. Default: 2
wayfinding.path-stroke-max-width	no		1.1	This is a decimal value indicating how thick the stroke is for the default wayfinding path at the map's maximum zoom level. Default: 40
wayfinding.path-stroke-color	no		1.0	A HEX color to draw the wayfinding path. Default: NULL.
wayfinding.path-stroke-alpha	no		1.0	A decimal value from 0 to 1 to indicate how transparent the

				default wayfinding path is. Default: NULL.
wayfinding.path-arrow-fill-color	no		1.1	This is a HEX color to draw the arrows on the wayfinding path. Default: #4688F1
wayfinding.path-arrow-stroke-color	no		1.1	This is a HEX color of the outline around the arrows drawn on the wayfinding path.Default: #FFFFFF
wayfinding.path-arrow-size	no		1.1	This is a decimal value indicating how large the arrows on the wayfinding path show up. The value should be in METERS. Default: 1.5
wayfinding.highlighted-path-stroke-width	no		Deprecated 1.1	A decimal value indicating how thick the stroke is for the highlighted section of the wayfinding path. Default: NULL.
wayfinding.highlighted-path-stroke-min-width	no		1.1	This is a decimal value indicating how thick the stroke is for the highlighted section of the wayfinding path at the map's minimum zoom level. Default: 2
wayfinding.highlighted-path-stroke-max-width	no		1.1	This is a decimal value indicating how thick the stroke is for the highlighted section of the wayfinding path at the map's maximum zoom level. Default: 40
wayfinding.highlighted-path-stroke-color	no		1.0	A HEX color to draw a highlighted section of the wayfinding path. Default: NULL.
wayfinding.highlighted-path-stroke-alpha	no		1.0	A decimal value from 0 to 1 to indicate how transparent the highlighted section of the wayfinding path is. Default: NULL.
wayfinding.highlighted-path-arrow-fill-color	no		1.1	This is a HEX color of the arrows drawn in a highlighted section of the wayfinding path. Default: #4688F1
wayfinding.highlighted-path-arrow-stroke-color	no		1.1	This is a HEX color of the outline around the arrows drawn in a highlighted section of the

				wayfinding path. Default: #FFFFFF
wayfinding.highlighted-path-arrow-size	no		1.1	This is a decimal value indicating how large the arrows in a highlighted section of the wayfinding path should be. The value should be in METERS. Default: 1.5

Possible Style Layer ID Patterns

This is a list of possible layer-IDs that can be styled 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 in which each layer would appear within the map. Layers with a higher *Order* value 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 (Google Maps or Apple Maps) 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].

Order	Layer-ID	Tile-type	Layer-type	Description
9	floor_walkways_[FLOOR]	vector	Polygon	Walkway polygons for the given [FLOOR].
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]	Raster	n/a	Raster map tiles for the given floor [FLOOR].
21	building_outlines_[BUILDING]	Raster	n/a	Raster map tiles for the given building [BUILDING].
22	floor_labels_[FLOOR]	All	Label	Labels for the given [FLOOR].
23	floor_icons_[FLOOR]	All	Icon	Icons for the given [FLOOR].
24	building_labels_[BUILDING]	All	Label	These are the labels for a given [BUILDING] that are displayed when no active floors in that building are shown.