DWG package requirements in Azure Maps | Microsoft Docs

farah-alyasari

# DWG package requirements

The Azure Maps Conversion API allows you to convert a DWG design package, for a single facility, into map data set. This article helps you understand the DWG package requirements for the Conversion API.

## Prerequisites

You may choose any CAD software to produce your DWG package.

You’ll also need to acquaint yourself with the following terms before we explain the DWG package requirements.

|  |  |
| --- | --- |
| Term | Definition |
| Layer | An AutoCAD DWG layer |
| Level | An area of a building at a set elevation. For example, the floor of a building |
| Xref | A DWG file attached to the primary drawing as an external reference |
| Feature |  |

## Package structure

A DWG package consists of DWG files and a manifest file. The DWG files can be organized in any way inside the zip folder, but the manifest file must live at the root directory of the folder. The files in the package must define a single facility, and be zipped in a single folder. The next sections detail the requirements for the DWG files, manifest file, and the content of these files.

## DWG files requirements

A single DWG file is required for each level of the facility. And all data for a given level must be contained in a single DWG file. Any external references (xrefs) must be bound to the to the parent drawing. Additionally, each DWG file:

* Must define the *Exterior* and *Unit* layers. It’s recommended that each level defines the following optional layers: *Wall*, *Door*, *UnitLable*, *Zone*, and *ZoneLabel*.
* Must be aligned to the same latitude and longitude as the origins of drawings for all DWG files of the facility
* Must not contain features from multiple levels
* Must not contain features from multiple facilities

The Azure Maps Conversion Service can extract the following feature classes from the DWG files:

* Levels
* Units
* Zones
* Openings
* Walls

A DWG layer must contain features of a single class. Classes cannot share a layer. For example, units and walls cannot share a layer. Also, each layer must be in the same orientation, as the other layers.

The table below outlines the supported entity types for each layer. Non-supported entity types will be ignored by the layer.

|  |  |
| --- | --- |
| Layer | Supported entity type |
| Exterior | Polygon, PolyLine (closed) |
| Units | Polygon, PolyLine (closed) |
| Walls | Polygon, PolyLine (closed) |
| Doors | Polygon, PolyLine, Line, CircularArc, Circle |
| Zones | Polygon, PolyLine (closed) |
| UnitLabel | Text (single line) |
| ZoneLabel | Text (single line) |

The next sections detail the requirements for each layer.

### Exterior layer

The DWG file for each level must contain a layer to define that level’s perimeter. This layer is refereed to as the exterior layer.

Regardless of how many entities are drawn in the exterior layer, the resulting facility data set will contain only one level feature for each DWG file. Additionally, the exterior layer:

* Must contain at least one closed PolyLine, which defines the exterior perimeter of the building at that level
* Must not contain any self-intersecting geometry
* Must not contain multiple overlapping geometries

If the layer contains multiple overlapping PolyLines, then the PolyLines will be dissolved into a single Level feature. Alternatively, if the layer contains multiple non\_overlapping PolyLines, the resulting Level feature will have a multi-polygonal representation.

### Units layer

The DWG file for each level should define a layer containing units. Units are navigable spaces in the building, such as offices and hallways. The units layer should adhere to the following requirements:

* Units must be drawn as closed PolyLines
* Units must fall inside the bounds of the facility exterior perimeter
* Units must not overlap
* Units must not contain any self-intersecting geometry

A unit can be named by creating a text object in the *unitLabel* layer and placing the object inside the bounds of the unit. See the [UnitLabel layer](#UnitLabel-layer) section for more details.

### Walls layer

The DWG file for each level may contain a layer that defines the physical extents of walls, columns, and other building structure.

* Walls must be drawn as closed PolyLines
* The wall layer(s) should only contain geometry that’s interpreted as building structure

### Doors layer

A DWG layer containing doors may be included in the package. To create a DWG layer for doors, each door must overlap the edge of a unit from the unit layer.

Doors won’t be rendered on the resulting map, as drawn in the CAD software. However, the doors will drawn according to the Azure Maps styling rules for the opening features.

### Zones layer

The DWG file for each level may contain a zone layer that defines the physical extents of zones. A zone layer:

* Must be drawn as closed PolyLines
* May overlap
* May fall inside or outside the facility’s exterior perimeter

A zone can be named by creating a text object in the *zoneLabel* layer, and placing the text object inside the bounds of the zone. See the [ZoneLabel layer](#ZoneLabel-layer) for more details.

### UnitLabel layer

The DWG file for each level may contain a unit label layer that adds a name property to units extracted from the unit layer. Units with a name property can have additional details specified in the manifest file.

* Unit labels must be a single line text entities.
* Unit labels must fall inside the bounds of their unit.
* Units must not contain multiple text entities in the unit labels layer.

### ZoneLabel layer

The DWG file for each level may contain a zone label layer that adds a name property to zones extracted from the zone layer. Zones with a name property can have additional details specified in the manifest file.

* Zones labels must be single line text entities.
* Zones labels must fall inside the bounds of their zone.
* Zones must not contain multiple text entities in the zone labels layer.

## Manifest file requirements

The zip folder must contain a manifest file at the root level of the directory, and the file must be named **manifest.json**. As indicated by the name, the manifest file is a JSON text file. It defines DWG file names, georeferencing information, and facility details.

The DWG file names must match the layer names. The file paths, in the **building\_levels** object of the manifest file, must be relative to the root of the zip file. Only the files identified by the manifest will be ingested by the Conversion API. Files that aren’t properly listed in the manifest will be ignored.

Although there are requirements when using the manifest objects, not all objects are required. The table below shows the required and the optional objects for version 1.1 of the Conversion API.

|  |  |
| --- | --- |
| Object | Required |
| directoryInfo | true |
| buildingLevels | true |
| georeference | true |
| dwgLayers | true |
| unitProperties | false |
| zoneProperties | false |

The next sections detail the requirements for each objects.

### directoryInfo

|  |  |  |  |
| --- | --- | --- | --- |
| Property | type | Required | Description |
| name | string | true | Name of building |
| streetAddress | string | false | Address of building |
| unit | string | false | Unit in building |
| locality | string | false | Name of an area, neighborhood, or region. Locality isn’t part of the mailing address. For example, “Overlake” or “Central District”. |
| adminDivisions | JSON Array of strings | false | An array containing address designations (Country, State, City) or (Country, Prefecture, City, Town). Use ISO 3166 country codes and ISO 3166-2 state/territory codes. |
| postalCode | string | false | Mail sorting code |
| hoursOfOperation | string | false | Adheres to the [OSM Opening Hours](https://wiki.openstreetmap.org/wiki/Key:opening_hours/specification) format |
| phone | string | false | Phone number associated with the building, and it must include country code |
| website | string | false | Website associated with the building, and it must begin with http or https |
| nonPublic | bool | false | Flag specifying if the building is open to the public. |
| anchorLatitude | numeric | false | Latitude of facility anchor (pushpin) |
| anchorLongitude | numeric | false | Longitude of facility anchor (pushpin) |
| anchorHeightAboveSeaLevel | numeric | false | Height of facility’s ground floor above sea level, in meters |
| defaultLevelVerticalExtent | numeric | false | Default height (thickness) of a level of this facility to use when a level’s verticalExtent is undefined |

### buildingLevels

The buildingLevels object contains a JSON array of buildings levels.

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Type | Required | Description |
| level\_name | string | true | Descriptive level name. For example: Floor 1, Lobby, Blue Parking, Basement, and so on. |
| ordinal | integer | true | A signed integer. Zero is nominally ground level, but this format isn’t strictly enforced. Ordinal is used to determine the vertical order of levels. |
| heightAboveFacilityAnchor | numeric | false | Level height above ground floor in meters |
| verticalExtent | numeric | false | Floor to ceiling height (thickness) of the level in meters |
| filename | string | true | File system path of the CAD drawing for a building level. It must be relative to the root of the building’s zip folder. |

### georeference

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Type | Required | Description |
| lat | numeric | true | Decimal representation of degrees latitude at the facility drawing’s origin |
| lon | numeric | true | Decimal representation of degrees longitude at the facility drawing’s origin |
| angle | numeric | true | The angle from the desired orientation of the building on a map to the orientation of the building in the DWG file. The angle is measured clockwise and in degrees. |

### dwgLayers

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Type | Required | Description |
| exterior | Array of strings | true | Names of layer(s) that define the exterior building profile |
| unit | Array of strings | true | Names of layer(s) that define units |
| wall | Array of strings | false | Names of layer(s) that define walls |
| door | Array of strings | false | Names of layer(s) that define doors |
| unitLabel | Array of strings | false | Names of layer(s) that define names of units |
| zone | Array of strings | false | Names of layer(s) that define zones |
| zoneLabel | Array of strings | false | Names of layer(s) that define names of zones |

### unitProperties

The unitProperties object contains a JSON array of unit properties.

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Type | Required | Description |
| unitName | string | true | Name of unit to associate with this unitProperty record. This record is only valid when a label matching unitName is found in the unitLabel layer(s) |
| categoryName | string | false | Category Name. For a complete list of categories, refer to [space categories](https://aka.ms/pa-indoor-spacecategories). |
| navigableBy | string[] | false | Indicates the types of navigating agents that can traverse the unit. For example, “pedestrian”. This property will inform the wayfinding capabilities. The permitted values are: pedestrian, wheelchair, machine, bicycle, automobile, hired\_auto, bus, railcar, emergency, ferry, boat, and disallowed. |
| routeThroughBehavior | string | false | The route through behavior for the unit. The permitted values are: disallowed, allowed, and preferred. |
| occupants | directoryInfo[] | false | List of occupants for the unit |
| nameAlt | string | false | Alternate Name |
| nameSubtitle | string | false | Subtitle |
| addressRoomNumber | string | false | Room/Unit/Apartment/Suite number of the unit |
| verticalPenetrationCategory | string | false | Vertical Penetration Category Name. For a complete list of categories, refer to [space categories](https://aka.ms/pa-indoor-spacecategories). When this property is defined, the resulting feature will be Vertical Penetration (VRT), meaning it can be used to navigate to other VRT features in levels above or below it. |
| verticalPenetrationDirection | string | false | If verticalPenetrationCategory is defined, optionally define the valid direction of travel. The permitted values are: low\_to\_high, high\_to\_low, both, and closed. |

### The zoneProperties object

The zoneProperties object contains a JSON array of zone properties.

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Type | Required | Description |
| zoneName | string | true | Name of zone to associate with zoneProperty record. This record is only valid when a label matching zoneName is found in the zoneLabel layer |
| categoryName | string | false | Category Name. For a complete list of categories, refer to [zone categories](https://aka.ms/pa-indoor-spacecategories). |
| zoneNameAlt | string | false | Alternate Name |
| zoneNameSubtitle | string | false | Subtitle |

### A sample manifest file

{  
 "version": "1.0",  
 "directoryInfo": {  
 "name": "Contoso Building A",  
 "streetAddress": "12345 NE 67th St",  
 "unit": "",  
 "locality": "",  
 "postalCode": "98052",  
 "adminDivisions": ["Contoso city", "WA", "United States"],  
 "hoursOfOperation": "Mo-Fr 08:00-17:00 open",  
 "phone": "1 (234) 567-8910",  
 "website": "www.contoso.com",  
 "nonPublic": true  
 },  
 "buildingLevels": {  
 "levels": [{  
 "levelName": "US\_STDB\_X1",  
 "ordinal": -1,  
 "heightAboveGround": -4.0,  
 "filename": "./US\_STDB\_X1.dwg"  
 },  
 {  
 "levelName": "US\_STDB\_01",  
 "ordinal": 0,  
 "heightAboveGround": 0.0,  
 "filename": "./US\_STDB\_01.dwg"  
 },  
 {  
 "levelName": "US\_STDB\_02",  
 "ordinal": 1,  
 "heightAboveGround": 3.0,  
 "filename": "./US\_STDB\_02.dwg"  
 },  
 {  
 "levelName": "US\_STDB\_03",  
 "ordinal": 2,  
 "heightAboveGround": 6.0,  
 "filename": "./US\_STDB\_03.dwg"  
 },  
 {  
 "levelName": "US\_STDB\_04",  
 "ordinal": 3,  
 "heightAboveGround": 9.0,  
 "filename": "./US\_STDB\_04.dwg"  
 }  
 ]  
 },  
 "georeference": {  
 "lat": 47.636152,  
 "lon": -122.132600,  
 "angle": 89.5  
 },  
 "dwgLayers": {  
 "exterior": ["GROS$"],  
 "unit": ["RM$"],  
 "wall": ["A-WALL-EXST", "A-WALL-CORE-EXST", "A-COL-EXST"],  
 "door": ["A-DOOR-EXST", "A-DOOR-CORE-EXST"],  
 "unitLabel": ["RM$TXT"]  
 },  
 "unitProperties": [{  
 "unitName": "3C00",  
 "categoryName": "office",  
 "navigableBy": "pedestrian",  
 "routeThroughBehavior": "preferred",  
 "unitInfo": {  
 "unit": "3C",  
 "name": "Contoso Team A",  
 "website": "https://contoso.com",  
 "phone": "1 (235) 567-9871"  
 }  
 }]  
}

## Next steps

Once your DWG package meets the outlined requirements, you may use the Conversion API to convert the DWG file to a map data set. Then, you can use the data set to generate an indoor map using the Indoor Maps module. Learn more about using the Indoor Maps module by reading the following articles:

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

[Indoor Maps data management](indoor-data-management.md)

[Indoor Maps dynamic styling](indoor-map-dynamic-styling.md)