DWG package requirements in Azure Maps | Microsoft Docs

farah-alyasari

# DWG package requirements

Files that define a single building must be packaged in a single zip folder, also known as the DWG package. The zip folder must contain DWG files and a manifest file. The next sections outline the requirements for the DWG files, manifest file, and the format of the content inside these files. Once your DWG package meets the requirements, you can use the DWG Conversion API to convert your DWG design into a map data set. The DWG Conversion API is provided by the Indoor Maps module from Azure Maps.

## Prerequisites

The DWG package is used to store two-dimensional design data or three-dimensional design data, and the metadata for the design. You may choose any CAD software to produce your DWG package. It’s also recommended that you familiarize yourself with the following terms:

|  |  |
| --- | --- |
| Term | Definition |
| Layer | An AutoCAD DWG layer |
| Level | An area of a building at a set elevation. Levels aren’t necessarily contiguous. For example, buildings with multiple towers have levels at different heights |
| Xref | A DWG package attached to the primary drawing as an external reference |

## DWG file requirements

The DWG files may be organized in any way inside the zip folder, but the content of the DWG files must adhere to the following rules:

* Each building level must be stored in one, and only one, DWG file
* Each building level must define the following layers: Exterior and Unit
* If a drawing of a level contains external references, then the drawing can be merged with these references using the bind command
* The origins of drawings for multiple levels must align to the same latitude and longitude
* Each level must have the same orientation as all the other levels
* All latitude and longitude coordinates must use the WGS84 reference coordinate system
* Geometry outside the building level outline will be ignored

In addition to the required layers, it’s recommenced that each building level defines the following layers: *Wall*, *Door*, *UnitLable*, *Zone*, and *ZoneLabel*.

Text objects, in any layer, won’t render on the map. However, text objects that are placed inside the feature geometry will be attached as attributes to the geometry, and then they’ll be exposed via a query layer.

### Exterior layer

The DWG file for each level should contain a layer that defines that level’s perimeter. It’s commonly known as the exterior layer or the outline layer, and the following requirements apply for the exterior layer:

* The exterior layer must contain a single closed polyline, which defines the exterior perimeter of the building at that level
* The exterior layer shouldn’t contain any self-intersecting geometry.
* The exterior layer shouldn’t contain multiple overlapping geometries. If the exterior layer happens to contain multiple overlapping geometries, then a union operation will be performed on all geometries in the layer.

### Units layer

The DWG file for each level should define a layer containing units. A unit is a room or a building space. Name a unit by creating a text object in the *unitLabel* layer and place the object inside the bounds of the unit. The units layer should adhere to the following requirements:

* Units should be drawn as closed polylines
* Units must not overlap
* Units shouldn’t contain any self-intersecting geometry
* Units should fall inside the bounds of the building outline for a given layer

### Walls layer

The DWG file for each level may contain a layer that defines the physical extents of walls and other building structure. Walls and structure may be spread across multiple layers, but layers containing walls and structure shouldn’t contain any other geometry. DWG files containing wall layers should adhere to the following conditions:

* Walls should be drawn as closed polylines
* The wall layer(s) should only contain geometry that’s interpreted as building structure
* The wall layer(s) must not contain self-intersecting geometry

[!Note] In the AutoCAD software, the polyline 'closed' property does not need to be set to 'Yes', but the first and last vertices of the polyline must be coincident.

### Doors layer

A DWG layer containing doors may be included in the package. To create a DWG layer for doors, you must adhere to the following conditions:

* Each door must overlap the edge of a unit from the unit layer
* Doors may be drawn using any 2D primitives, such as a line, polyline, or arc

Doors won’t be rendered on the indoor map, but the doors layer will be used to give information to the **Indoor WayFinding** algorithm.

### Zones layer

The DWG file for each level may contain a zone layer that defines the physical extents of zones. Name a zone by creating a text object in the *zoneLabel* layer, and place the text object inside the bounds of the zone. Below are the conditions for the zones layer:

* Zones must be drawn as closed polylines
* Zones may overlap
* Zones may fall inside or outside the level boundary

## Manifest file requirements

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

The next sections define the objects of the manifest file and the properties of these objects. Properties marked as required must be defined in the manifest file. Properties marked as not required are optional, but it’s recommended to define these properties to add details and clarity to the map.

### The directoryInfo object

|  |  |  |  |
| --- | --- | --- | --- |
| 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 string | 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. |
| pushpinLocation | {Lat,Lon} | false | Location to use when representing the building as a point on the map |

### The buildingLevels object

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. |
| heightAboveGround | null or numeric | false | Level height above ground 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. |

### The georeference object

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Type | Required | Description |
| lat | numeric | true | Decimal representation of degrees latitude at the building origin to six or more digits of precision |
| lon | numeric | true | Decimal representation of degrees longitude at the building origin to six or more digits of precision |
| 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. It’s measured clockwise in degrees. |

### The dwgLayers object

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

### The unitProperties object

The unitProperties object contains a JSON array of unit properties.

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Type | Required | Description |
| unitName | string | true | Name of unit to associate with unitProperty record. This functionality is only supported when a unitLabel layer is used to assign names to units. |
| 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 object will inform the wayfinding capabilities. Currently this property isn’t utilized by Private Atlas. |
| routeThroughBehavior | string | false | The route through behavior for the unit. The following are the allowed values for the navigable\_by subtype: 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). |
| verticalPenetrationDirection | string | false | If verticalPenetrationCategory is defined, optionally define the valid direction of travel. The following are the allowed values for the navigable\_by subtype: 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 functionality is only supported when a zoneLabel layer is used to assign names to units |
| categoryName | string | false | Category Name. For a complete list of categories, refer to [space 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:

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

[!div class=“nextstepaction”] [Indoor Maps data management](indoor-data-management.md)

[!div class=“nextstepaction”] [Indoor Maps dynamic styling](indoor-map-dynamic-styling.md)