# Custom Rendering

In OsmAnd, you have the ability to change the look and feel of the maps as they are drawn (only in offline, vector rendering of course). This includes changing:

* colors
* line thickness
* line borders/dashed lines
* at what zoom level icons/lines are visible

and more!

OsmAnd uses an xml file to determine how to shows this information, the built in ones can be found in the git repository:

<https://github.com/osmandapp/OsmAnd-resources/tree/master/rendering_styles>

The render.xml files (so-called “depends” renderers) use inheritance. So if you e.g. don’t like the colour of primary roads in the default renderer (defalt.render.xml), you can create your own CUSTOM.render.xml:

A template to write a depends renderer is provided as file Depends-template.render.xml. Create a local copy named e.g. CUSTOM.render.xml, Make sure in line 1the dependency is set to “default.render.xml” . The following line

<renderingConstant name="primaryRoadColor" value="#000000"/>

would set the primary road colours to black (#000000).

OsmaAnd reads from top to bottom of parent xmls, then top to bottom of child (depends) xmls. It will keep the last setting that it reads. So in the example above, it will set the primary road colour to #FFAA80, then when it reads the child xml, it will set it to #00000. So as it reads down the page, it will overwrite existing values.

You can cascade render.xml dependencies, i.e. a “4wd.render.xml” could touring-view.render.xml, which in turn depends on default.

As mentioned, when creating your own dependant render.xml, start with default\_template.render.xml. Rename a copy appropriately (keeping “.render.xml” !) and make the desired changes. When you are done with your new scheme, simply put the file in the existing“renderig” sub-folder on your phone (under the osmand data folder) and completely restart OsmAnd .

## Changing Icons

In short you cannot really.

The icons are compiled into the program, so if you wish to add another one, you need to create a pull request into the git icon repository. It will then be included.

## Add new Features to the Map

If you want to add something that isn’t on the map, you need to check that it is actually compiled into the .obf database. There is a finite list and not everything can be included by default. The list of features created by default can be found in the **rendering\_types.xml.**

<https://github.com/osmandapp/OsmAnd-jni/blob/master/java/net/osmand/osm/rendering_types.xml>

To add in new features (for example I have added **4wd\_only** tracks to my own maps), create your own rendering\_types.xml and save it.

Open map creator, goto settings and add the path to your new rendering\_types.xml. Now it will render from that file instead of the default.

Edit rendering\_types.xml and add the appropriate lines, I have added the 4wd\_only tags as shown below:

<category name="4wd\_grade">

<type tag="4wd\_grade" value="green" minzoom="10" />

<type tag="4wd\_grade" value="blue" minzoom="10" />

<type tag="4wd\_grade" value="black" minzoom="10" />

<type tag="4wd\_grade" value="red" minzoom="10" />

</category>

You also have to make sure that it is rendered in the correct order, inbetween the <order></order> tags in your render.xml, add the appropriate lines, ie for my new tracks, I have copied the highway:primary settings.

<order>

<filter layer="1" tag="4wd\_only" value="green" order="87"/>

<filter tag="4wd\_only" value="green" order="57"/>

<filter layer="1" tag="4wd\_only" value="blue" order="87"/>

<filter tag="4wd\_only" value="blue" order="57"/>

<filter layer="1" tag="4wd\_only" value="black" order="87"/>

<filter tag="4wd\_only" value="black" order="57"/>

<filter layer="1" tag="4wd\_only" value="red" order="87"/>

<filter tag="4wd\_only" value="red" order="57"/>

</order>

## Layout of render.xml

<renderingStyle name="" depends=" " defaultColor="#f1eae4" version="1">

*<!—These are “Global” variables, which can be referenced throughout the file -->*

<renderingProperty>

<renderingAttribute>

<renderingConstant>

<order>

*<!-- This section determines the order in which objects are rendered. Higher values will be on top! -->*

</order>

<text>

*<!—This determines how text object are displayed -->*

</text>

<point>

*<!—This determines how point objects, which typically have an icon associated with them are displayed -->*

</point>

<polygon>

*<!—This determines how polygon objects are displayed -->*

</polygon>

<line>

*<!—This is typically the most imnportant and complex section. It determines roads, footpaths, coastlines etc. Basically everything that is a line goes into this section! -->*

</line>

</renderingStyle>

## Tag Types

renderingProperty

renderingAttribute

renderingConstant

There are two types of tags, Attributes and Values.

An **attribute** is used to describe the object or the state of the app, ie its tag/value, whether the app is in car, pedestrian or cycle mode, what the zoom is etc.

A **value** allows you to set how the object is displayed, it’s colour, icon, stroke width etc.

|  |  |  |  |
| --- | --- | --- | --- |
| Attr | appMode= | If the app is in the defined appMode, the filter will be applied. | * pedestrian * car * cycle |
| Value | area= | Used to set whether an object has an area, ie on polygons | * true * false |
| Value | attrColorValue= | Used to set the colour of renderingAttribute | * #*xxxxxx* |
| Value | attrIntValue= | Used to set an integer value of a renderingAttribute | * Any integer |
| Value | color\_2= | Second colour value | * #*xxxxxx* |
| Value | color= | Colour to assign to the referenced tag/value. | * #*xxxxxx* |
| Attr | contourLines= |  |  |
| Attr | cycle= |  | * True * false |
| Value | icon= | The icon associated with the tag/value |  |
| Value | layer= |  |  |
| Attr | maxzoom= | The maximum zoom that the tag/value will be displayed. Can be used in conjuction with minzoom= to set a range | * 1 to 20 |
| Attr | minzoom= | The minimum zoom that the tag/value will be displayed. Can be used in conjuction with maxzoom= to set a range | * 1 to 20 |
| Value | name= |  |  |
|  | nameTag= |  |  |
| Attr | nightMode= |  |  |
| Attr | noPolygons= |  |  |
| Value | order= | The order in which an object is drawn | * Any integer |
| Value | pathEffect= | This is used to create dashed lines. It must be even, and the numbers indicate length of the on and off sections. It always begins with the On section. | * On\_Off * 5\_5 * 5\_5\_2\_2 * 2\_2\_5\_2 * etc |
| Attr | point= | True is the object is a point object (ie one node only) | * true * false |
| Value | possibleValues= |  |  |
|  | roadColors= |  |  |
|  | shadowColor= |  |  |
|  | showRoadMaps= |  |  |
| Value | strokeWidth\_2= |  |  |
| Attr | tag= | Tag name, as used in OSM. |  |
|  | test= |  |  |
| Value | textBold= |  |  |
| Value | textColor= |  |  |
|  | textDy= |  |  |
|  | textHaloRadius= |  |  |
|  | textMinDistance= |  |  |
| Value | textOnPath= |  |  |
|  | textOrder= |  |  |
|  | textShield= |  |  |
| Value | textSize= |  |  |
|  | textWrapWidth= |  |  |
|  | type= |  |  |
| Attr | value= | Value of the OSM tag |  |

## Group

## filter

the filter tag allows you to determine what is being affected (ie tag/value) as well as the attribute it should adopt, and in what situations.

Ie, I want my track (4wd\_only:black) to be black, and visible from zoom level 10

<filter tag="4wd\_only" value="black" color="#000000" minzoom="10" strokeWidth="2"/>

This is the simplest form of the filter tag, and is used extensively.

### Nesting filter

Ie, I want my track (4wd\_only:black) to be black, and visible from zoom level 10, but at zoom 14, I want the path to be wider.

To do this I will nest the filters.

<filter tag="4wd\_only" value="black" color="#000000"/>

<filter minzoom="10" maxzoom="13" strokeWidth="2" />

<filter minzoom="14" maxzoom="20" strokeWidth="10"/>

</filter>

The first filter sets the tag/value and the common attributes (ie colour). The nested filters set the individual attribute, in this case dependant on the zoom levels. Other attributes can be used, ie appMode.

## groupFilter

The group Filter will set everything in the group to the defined attribute.

Ie everything in the group is to have a round cap between zooms 5 and 20.

<groupFilter cap="ROUND">

<filter minzoom="5" maxzoom="20" strokeWidth="1.6"/>

</groupFilter>

The groupFilter can also use attributes to display the object differently in certain situations.

Ie if the highway has text on it, change the size of the text depending on the zoom.

<group>

<filter tag="highway" value="motorway" nameTag="" textOrder="6"/>

<filter tag="highway" value="motorway\_link" nameTag="" textOrder="7"/>

<filter tag="highway" value="trunk" nameTag="" textOrder="6"/>

<filter tag="highway" value="trunk\_link" nameTag="" textOrder="7"/>

<filter tag="highway" value="primary" nameTag="" textOrder="7"/>

<filter tag="highway" value="primary\_link" nameTag="" textOrder="8"/>

<filter tag="highway" value="secondary" nameTag="" textOrder="8"/>

<filter tag="highway" value="secondary\_link" nameTag="" textOrder="9"/>

<groupFilter textOnPath="true">

<filter minzoom="13" maxzoom="13" textSize="12"/>

<filter minzoom="14" maxzoom="14" textSize="13"/>

<filter minzoom="15" maxzoom="15" textSize="14"/>

<filter minzoom="16" maxzoom="16" textSize="15"/>

<filter minzoom="17" maxzoom="17" textSize="17"/>

<filter minzoom="18" textSize="19"/>

</groupFilter>

</group>