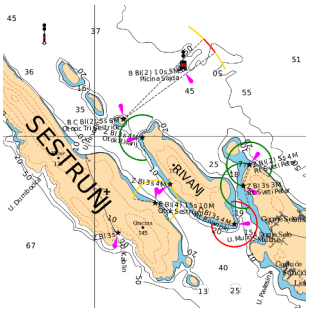

Smrender

A Rule-based Renderer for OSM Data

Bernhard R. Fischer

December 21, 2020

Contents

1. Release Notes

This document describes the current (2015/08/22) version of *Smrender* which is tagged as version 4.0. The document corresponds to the internal SVN revision number 1854 of *Smrender*. Starting with version 3.0 *Smrender* supports libcairo[1] instead of libgd as its graphical library. There are some major changes after SVN revision 1240, thus this document does not apply to revisions 1240 and earlier. The format of the ruleset changed and are **not** compatible.
Unfortunately, this documentation is not complete yet, but I will continue to work on it.
Smrender contains several functions which are experimental. Those functions are namely the auto-rotation and the polygon-size dependent captions (see Section 6.1.1).

2. Name

Smrender is a universal rule-based rendering engine for OSM data. Because *Smrender* is a very generic and flexible OSM processing engine, it may be used for different tasks such as data filtering or data modification.

3. Synopsis

smrender [OPTIONS] [window]

4. Description

Smrender reads an OSM file and applies a set of rules to this input data to create an output image. The input is an OSM file and a set of files containing the rule set. Supported output formats are currently PDF, SVG, PNG, and KAP/BSB[2] having the desired resolution and density. Additionally, some other output files may be generated depending in the rule set. The latter is explained later.
The primary goal of *Smrender* is to create a sea chart which is well-suited for print-out on paper. Nevertheless, it is a universal rendering engine and may be used for different tasks.
The input file should be an OSM/XML file as defined by the OSM standard. The file is required to be well-formed in that sense because *Smrender* itself does no XML validation, thus the rendering process might fail if the file is not well-formed. The data should also be complete. This means that it should contain all nodes to which is referred by the ways. *Smrender* will remove nodes from ways which are missing.
The rules are also defined in OSM format (see Section 5). The rules are applied iteratively in a loop depending on their *version*. Within the loop, *Smrender* always applies first all relation rules, then way rules, and then all node rules of the same version. All rules of the same version are applied in the order of their *id*.
Invisible objects are ignored by the renderer. Invisible objects are such which have set the attribute visible="false". If objects have no such attribute *Smrender* sets it to "true" by default.

4.1. Rendering Window

Smrender renders an area which is specified by the window. It is a compound argument as defined below.

[1] http://www.cairographics.org/
[2] http://opencpn.org/ocpn/kap_format

```
<window>      := <center> | <bbox>
<bbox>        := <left lower>:<right upper>
<left lower>  := <coords>
<right upper> := <coords>
<center>      := <coords> ':' <size>
<coords>      := <dec_coords> | <naut_coords>
<dec_coords>  := <lat> ':' <lon>
<naut_coords> := ( <naut_lat> ':' <naut_lon ) |
                 ( <naut_lon> ':' <naut_lat> )
<size>        := <scale> | <length> 'd' | <length> 'm'
```

The area to be rendered is either specified by a center point of the area or a bounding box.
If it is chosen to use a center point specification then lat and lon set the center coordinates in latitude and longitude in degrees in WGS84 reference system. Although it can be any valid coordinate, it is suggested to choose an "even" value rounded to 10 minutes (e.g. 43.666667 which is 43° 40').
Alternatively, the coordinates can be given in nautical notation which is dd C mm.m where dd is degrees as integer value, C is one of 'N', 'S', 'E', or 'W', and mm.m are minutes, for example 43N40. In case of nautical notation, *Smrender* automatically detects which of the the values is the latitude and which is the longitude dependent on the character used for the cardinal direction.
Length defines the length of the mean latitude (parallel) in degrees if 'd' is appended or in nautical miles if 'm' is appended. Alternatively, the scale of the chart can be specified. *Smrender* calculates the size of the area to meet the scale. Obviously, this depends on the size of the output image. The height (the length of the mean longitude) is calculated automatically by *Smrender* in such a way that the output image is projected correctly using a *Mercartor projection*. The height depends on the size of the output image (page format).

If it is chosen to use a bounding box specification then the coordinates of the left lower (southern western) and the right upper (northern eastern) point shall be specified. Because of the projection the bounding box may not necessarily fit to the page dimension in which case *Smrender* will automatically resize the bounding box to fit to the page. If a fixed bounding box is desired the option **-P** is needed with either the width or the height set to zero. In this case *Smrender* will calculate a page dimension which fits to the bounding box according to the projection.

If this argument is omitted The value 0:0:100000 is chosen as default and the option **-a** is set implicitly.

4.2. Options

-a *Smrender* usually processes just those nodes which are located within the window of rendering (see Section 4.1). This behavior is new with version 3.0. With this option set *Smrender* processes all nodes independently of their location.

-b color

This option allows to set a background color. The color may be define either as a color preset or an HTML-style definition (see Section 6.1.1). The default color is white.

Please note that some Shells (such as the bash) interpret the # character if the color is specified in HTML-style. In this case you have to put the color specification under single or double quotes, e.g. -b '#8080ff'.

-D *Smrender* outputs a lot of information to stderr to watch the progress of rendering. It uses the same log levels as defined for syslog (see syslog(3)). By default it logs all messages of the log level LOG_INFO and above. With this option also debug messages are printed. This may be useful if something goes wrong.

-d density

Set the density of the output image. The default value is 300 which is typically used for print-outs in good quality. This option does only apply if you produce pixel (raster) images such as PNG or JPG.

-f Filter data while loading the input file.

With this option a bounding box is used to load just those nodes and ways which are within the selection. The bounding box is 10% larger then the area which resumes from the selected window. This option is useful if huge input data sets are used, such as the planet file.

Please note that this option works only correctly if the nodes and ways are stored in that order in the input file (first all nodes and then all ways).

-g d[:t[:s]]

This option defines the distance d of the grid lines in minutes. The border of the chart (latitude and longitude axis) also depends on this setting. As it is usual for sea charts, there is a major and a minor axis scale, called ticks and subticks. The ticks are defined by t and the subticks are defined by s in minutes. For a correct result, t as well as d should be integral multiples of s.

Note that *Smrender* internally uses a precision of hundreds of a minute while doing the grid calculation. Thus, the smallest granularity for the grid parameters is 0.01 minutes.

If this option is omitted, *Smrender* will choose a grid setting dependent on the scale of the output image. For scales below 1:150000 it chooses 5:1:0.2, for scales below 1:250000 it chooses 10:1:0.25, and for scales above 1:250000 it is set to 20:2:0.5.

A grid may optionally be generated with the action grid. See Section 6.3.4.

-G Do not generate grid nodes/ways. *Smrender* actually does not render the grid directly onto the output image but rather generates regular OSM nodes and ways. These objects are then rendered in a way as it is defined by the regular rule set. All ways of the grid are tagged with grid=*. Part of the grid are also labels on the border showing degrees in latitude an longitude. The labels are nodes which are tagged with grid=text and the tag name=* containing the value.

-h Output the list of available options and a short description to stdout and exit.

-i file

This option specifies the name of the input file. If this option is omitted, *Smrender* reads from stdin.

-k file

With this option *Smrender* will write a KAP/BSB file to file which can be used with most navigation applications and GPS devices. See Section 9.2 for more details.

-K file

This is similar to option **-k** but it generates only a KAP/BSB header instead of a complete file. This is useful if you want to use an external tool such as imgkap to create the KAP file from the PNG.

-l Output page has landscape format rather than portrait, which is default. This option is used only in conjunction with option **-P** if a literal page format is used.

-o file

Set the filename of the output image. *Smrender* parses the string and chooses the output file dependent on the file extensions which can be one of .pdf, .png, or .svg. It defaults to PNG if no valid extension was found. If this option is omitted no image file will be generated.

-O file

Create output PDF file. This option is *deprecated*. See option **-o**.

-p projection

Select the chart projection. This could either be mercator or adams2, the 1st being the default value if the options is omitted.

-P fmt|geom

Select the page format of the output image. The format can be set either named format fmt which has a specific dimension or as geometry geom which contains the width and height of the page in millimeters in the format widthxheight. Fmt currently supports the values A4 up to A0. If this option is omitted, A3 format is selected by default.

If the area to be rendered is specified as bounding box (see Section 4.1) this will most probably colide with the dimension of the page because of the requirements of the projection in which case *Smrender* will resize the bounding box appropriately. If either the width or the height (but never both) is set to zero then *Smrender* will calculate the missing dimension according to the bounding box and the projection.

-m Disable memory mapping (see option **-M**).

-M This option is mandatory if the input file is larger than the amount of memory available on the rendering system. If the system has enough memory this option can be omitted. Using memory mapping probably is a little bit slower but heavily depends on the operating system. On Linux kernel 2.6.32 there is no significant difference in speed.

In any case, files which are larger then 2 GBytes are only supported on 64 bit operating systems.

This option is now on by default.

-N ofs

With this option *Smrender* adds ofs to all nodes and ways IDs as well as to the node references of the ways for all OSM files that are generated by *Smrender* by using the option **-w** or the action out() (see Section 6.3.2).

This options is mainly intended to renumber IDs from input files.

-n Some actions of *Smrender* generate new objects. Negative ids are added to those objects. If this data created by *Smrender* is fed to another OSM tool negative ids may cause problems. With this option all negative ids are output as positive values.

-r file

This option specifies the file name of the rules file. If it is omitted, *Smrender* expects the rules file to be named rules.osm. If file is set to none *Smrender* is run without rules.

If file is a directory *Smrender* will scan it and read all files which match "*osm". The files are read in alphabetical order, simply compared with strcmp(3). Thus, it is suggested to create symbolic links to the files in name it in an appropriate way similar to SysV init scripts.

If you need a more specific order of rule execution you should consider to use the version and/or id fields of the OSM objects (see Section 4).

-R file

This option may be used to save the rules again to another OSM file. This fill will be well-formed in terms of OSM/XML format and may be loaded into JOSM.

-s f

This option scales all images used in the img() function by the factor f (see Section 6.1.3). By default this is 1. This factor is applied additionally to a scaling defined in img().

-t title

This option generates a descriptional title node which can be rendered onto the final chart. It is as well used as a title for KAP files (see option **-k**).

-T z ':' dir [':' ftype]

Create tiles suitable for an online viewer into the directory dir. The zoom levels are specified by z which can be either a single value or a range, e.g. 10-15.

The optional parameter ftype sets the file type of the tile images which can be either png (which is default) or jpg.

-u With this option *Smrender* will output a set of URLs which are suitable for OSM data download for the area of rendering specified by the window.

-V With this option set, *Smrender* parses all arguments, and calculates and prints the rendering parameters to stderr. Then it exits immediately.

-v With this option *Smrender* will output the version information of *Smrender* and libcairo to stdout and exit immediately.

-w file

This option specifies the file name of an output OSM file. *Smrender* will dump all nodes/ways to this file that have been selected by the import process and all nodes/ways which have been generated during the rendering process.

These nodes/ways include the objects generated for the grid and also the close coastline ways. Thus the output depends on the options **-f**, **-C**, and **-G**.

If this option is omitted, no output will be generated.

5. Ruleset Definition

The rule set is also defined in OSM format. It contains nodes, ways, and relations together with tags. Nodes are considered to be rules for rendering nodes and ways are rules applied to ways. Each object (node, way, or relation) has a list of tags. These tags represent patterns which are matched against the tags of the objects which are to be rendered. The values of a tag's key (k="...") and/or value (v="...") may be either just a string which is directly matched in a case-sensitive manner or a special match operation (see Section 5.2). The match operations can be used for the key as well as for the value.
Each object has to have a special tag which defines the action that should be carried out in case of a match. This tag has the form _action_=*. The actions are described below in Section 5.3.
The match algorithm always applies all tags to match, and all of them have to match in order to execute the action. If just a single tag does not match, the node is skipped.

5.1. Order of Rule Execution

A typical ruleset consists of a set of node, way, and relation rules. It is important to understand the order in which the rules are executed. Since a page is flat and 2-dimensional, rules which are executed later may paint over those that have been executed before.
As defined by the OSM specification[3] all object types (nodes, ways, relations, also called *elements*) have common attributes. Beside the object type, *Smrender* uses the attributes version and id to determine the order of rule execution. All rules of the same version are rendered in the order of their id ascendingly within each type of object. This is repeated for each version ascendingly until the last rule (highest version and highes id). The following enumeration outlines the rendering order.

1. All objects are selected according to their version number, lowest number first.
2. All objects with the same version number are selected according to their type: all relations before ways before nodes.
3. All objects of the same type (and version) are chosen according to their id, lowest id first.

Rules with a version greater or equal to $2^{16} = 0\text{x}10000 = 65536$ are ignored. The number of iterations, i.e. the number of different versions is limited to MAX_ITER[4] which is defined in rdata.h. Run *Smrender* with option **-V** to see its value.
If the version attribute is missing it is set to 1 by default. All objects without id are numbered ascendingly in the order in which they occur in the rules file starting with some low negative value. The attribute visible can be set to either true (which is default if omitted) or false. Rules which are "invisible" are not executed. This can be used to enable or disable a rule by default and can further be used for conditional rendering (see enable_rule in Section 6.3.4).

5.2. Match Operations

Basically there are the four different match operations *string compare*, *regular expression match*, *greater than*, and *less than*. Additionally, all of them may be *inverted*, or *excluded*.

[3] `http://wiki.openstreetmap.org/wiki/Elements`
[4] It is currently (21^{st} of October 2014) defined to 64.

- *String compare* is the most basic match operation. It does an exact case-sensitive string compare. The following tag matches all objects which own the tag seamark:light_character=*.

```
<tag k='seamark:light_character' v=''/>
```

  The empty value v='' represents a wildcard match. It matches any string.

- *Regular expressions* are invoked by enclosing the expression within two slashes /.../ as it is usual in Perl and several other languages. The expression is interpreted as a POSIX extended regular expression (see manpages regex(3) and regex(7)). The following expression matches any object which is tagged with either highway=primary or highway=secondary.

```
<tag k='highway' v='/^(secondary|primary)$/'/>
```

- *Smrender* can interpret tag values as numerical values. Thus, it can do arithmetical comparisons which is *less than* ($<$) and *greater than* ($>$).

  The rule has to contain a number enclosed in square brackets. The direction of the brackets denotes the comparison operation; $[x]$ means that the value of the tag should be lower than x and $]x[$ matches if the tag value is greater than x. The tag value as well as x are always interpreted as decimal number with double precision.

  Square brackets are used deliberately instead of angle brackets to avoid possible conflicts with the XML format or buggy parsers.

  The following rule matches all objects whose seamark:light:range-value is greater than 7.5.

```
<tag k='seamark:light:range' v=']7.5['/>
```

- Match inversion is done by enclosing the match expression within exclamation marks. This inverts the match if and only if the expression without inversion would match. This means that if a tag would match the expression, the inversion would return "false" (no match). But it would not return "true" (match) if the expression would not match.

  The following expression would match all objects which have a tag with the key seamark:type but its value is neither landmark nor light_major.

```
<tag k='seamark:type' v='!/landmark|light_major/!'/>
```

- Match exclusion is denoted by enclosing the match expression within tildes. It does not match objects with certain tags, i.e. it matches the absence of the given tag. The following rule avoids matching of objects which have a tag seamark=*, i.e. all objects match which do not have such a seamark=* tag.

```
<tag k='~seamark~' v=''/>
```

5.3. Rule Actions

Smrender supports a number of powerful built-in actions which are carried out upon successful match. As already mentioned at the beginning of this Section, actions are defined simply with the tag _action_=*. The actions are actually function calls either within the code of *Smrender* or externally from a dynamic library. Thus, there is an unlimited range of extensibility of *Smrender*.

A number of parameters may optionally be passed to the function. The order of the parameters does not matter.
The following example shows an action which places an image at the position of a node.

```
<tag k='_action_' v='img:file=icons/Light_Minor.png'/>
```

The basic format of an action and basic values and types are defined as follows.

```
<action> := <ref> [ ':' <param> [ ';' <param> [ ... ] ] ]
<ref>    := <func> [ '@' <library> ]
<param>  := <tparam> | <bparam>
<tparam> := [ SEP ] <name> [ SEP ] '=' [ SEP ] <value> [ SEP ]
<bparam> := [ SEP ] <name> [ SEP ] '=' [ SEP ] <bool> [ SEP ]
<bool>   := 'yes' | 'no' | 'true' | 'false' | <num>
<num>    := any decimal number
SEP      := ' ' | '"'
<length> := <num> [ <udef> ]
<udef>   := <pageunit> | <nautunit>
<pageunit> := 'mm' | 'cm' | 'in' | 'px' | 'pt' | '"'
<nautunit> := 'degrees' | 'deg' | 'min' |
              'm' | 'km' | 'ft' | 'nm' | 'sm'
```

Every action consists of a symbol name ref which is used to find the appropriate function in *Smrender*or in a shared object[5] and an optional set of parameters. These are attribute/value pairs. The names of the attributes are case-sensitive. A special *type* exists which is the boolean type. It can be set to the case-insensitive strings 'yes', 'no', 'true', or 'false', or to any decimal number. 0 is interpreted as false all other values are considered to be *true*.
Smrender is shipped with a set of functions for rendering. Those are capable to place captions (Section 6.1.1), drawing and filling (Section 6.1.2), placing icons (Section 6.1.3), generating OSM files (Section 6.3.2), do some special purpose operations (Section 6.3.4), and calling external library functions (Section 6.3.1). The latter is thought to be a simple but powerful interface for third-party modules.
All functions are described in Section 6.

5.4. Measurement Units

Starting with revision 1851 *Smrender* knows about a set of standard measurement units.[6] There are two groups of units. The first one is related to the page to which the chart is rendered and the second group is related to reallity. Units referred to the page are listed in Table 1 and those referred to reallity are shown in Table 2.
Units are used in combination with a decimal value. As usual, the unit symbols are written behind the number and may be separated with zero or more white spaces.
Pixels do not have a fixed conversion factor. It depends on the pixel density and it is *only* important for raster image output. The density (dpi, dots per inch) is set with the command line option **-d**. The default value is 300.
Please note that *Smrender* internally operates vector-based and the suggested output format is PDF which is created as vector image as well. There are no pixels.

[5] Function libraries, on Unix-based systems .so files, on Mac OSX .dylib files, and on Windows .dll files.
[6] Please note that not all functions already know about that.

Symbol(s)	Descripton	Conversion factor
mm	Millimeters	= 0.03937" = 0.3527 pt
cm	Centimeters	= 10 mm
in or "	Inches	= 25.4 mm = 72 pt = 300 px
px	Pixel	= 0.0846 mm = 0,0033" (see below)
pt	Points	$=\frac{1}{72}$"

Table 1: Measurement units referred to the output page.

Symbol(s)	Descripton	Conversion factor
nm	Nautical miles	= 1852 m = 6076.12 ft
kbl	Cable length	= 185.2 m = 0.1 nm
' or min	Minutes	= 1 nm
° or deg	Degrees	= 60'
m	Meters	= 3,2808 ft
km	Kilometers	= 1000 m
ft	Feet	= 0.3048 m

Table 2: Measurement units referred to reallity.

Degrees and minutes are usually (if not specified different) referred to a great circle, i.e. 1° = 60' = 60 nm.

6. Rendering Functions

Generelly, a function does „something“ with the objects that match. The functions are called repeatedly for each objects with matching tags.
It can be distinguished between three types of functions:

1. Functions which directly produce graphics output. Those are also called *rendering primitives* and are described in Section 6.1.

2. Functions which manipulate OSM data, this is modifying or adding OSM elementes to the data. They are described in Section 6.2.

3. Special purpose functions which either control *Smrender* itself or do something else specific, e.g. producing OSM output files. These are described in Section 6.3.

6.1. Graphical Rendering Primitives

6.1.1. Captions

The action type cap is used to place a caption. If the action is carried out in a node-rule, the caption is placed at the node's position with the specified properties. The formal definition looks like the following.

```
<action>   := 'cap:' <param> [ ';' <param> ]
<param>    := <fontdef> | <angledef> | <aligndef> | <wcapopt>
              | <fopt> | <hide>
```

```
<fontdef>  := <fontname> | <fontsize> | <color> | <fopt>
              | <key>
<fontname> := 'font' '=' <string>
<fontsize> := 'size' '=' decimal value
<color>    := 'color' '=' <coldef>
<coldef>   := X11 color name or HTML specification "#xxxxxx".

<key>      := 'key' '=' <string>

<angledef> := <angle> | <anglekey>
<angle>    := 'auto' | decimal value
<anglekey> := 'anglekey' '=' <string>

<fopt>     := <foptname> '=' decimal value
<foptname> := 'weight' | 'phase' | 'size' | 'xoff' | 'yoff'

<wcapopt>  := <wcapname> '=' decimal value
<wcapname> := 'min_size' | 'max_size' | 'min_area' | 'auto_scale'

<aligndef> := <stat_algn> | <dyn_algn>

<stat_algn>:= <algn> | <halgn> | <valgn>
<algn>     := 'align' '=' <all_alg>
<all_alg>  := <lat_alg> | <lon_alg> | <diag_alg>
<lat_alg>  := 'north' | 'south'
<lon_alg>  := 'east' | 'west'
<diag_alg> := 'northeast' | 'southeast' | 'southwest'
              | 'northwest'
<halgn>    := 'halign' '=' <lon_alg>
<valgn>    := 'valign' '=' <lat_alg>

<dyn_algn> := <dalgn> | <dhalgn> | <dvalgn>
<dalgn>    := 'alignkey' '=' <all_alg>
<dhalgn>   := 'halignkey' '=' <lon_alg>
<dvalgn>   := 'valignkey' '=' <lat_alg>

<hide>     := 'hide' '=' <bool>
```

The parameters font, size, and key are mandatory, the others are optional.
If fontconfig is available, font is defined as specified by fontconfig (see fontconfig documentation). This is e.g. "font=serif:bold". If fontconfig is not available, font must be a full path to a TTF font file.
Size defines the size of the font in millimeters as a decimal value, for example "size=2.4".
Key specifies the key of the tag whose value should be printed. If a caption rule is applied to a node which does not have such a key, the rule simply does nothing. If the key is preceded by an asterisk '*', all letters are capitalized.
Valign and halign specify the alignment of the caption in respect to its center point which is given by the coordinates of the node. There is a horizontal alignment (halign) which could be either east or west and a vertical alignment (valign) which is one of north or south. If no alignment is

specified the caption will be centered.
Color defines the color in which the caption should be set. This is either an X11 color preset[7] such as "white", "yellow", "darkgreen" ..., or a color definition similar to the HTML standard. The color presets reflect the colors of traditional sea charts. The HTML-style color definition has the pattern `#[aa]rrggbb`. The values rr, gg, and bb reflect the RGB values as a hexadecimal number from 00 to ff. Optionally a transparency may be specified with aa. It ranges from 00 (opaque) to 7f which is absolute transparent. The most signifficant bit is always cleared, hence, setting values greater than 7f has no effect.
xoff and yoff specify an offset which is additionally added if the caption is *not* placed in the center point. The default value is set to size/2. This is because it looks better if the caption does not directly stick to its baseline or the left or right extremety of its bounding box but has a small distance. Sometimes, this is called *padding*.
The angle defines how the caption should be rotated. It is given as usual in trigonometrics which is degrees counterclockwise from 0 to 360 being 0 the regular left-to-right orientation.[8]
Alternatively, the angle may be set to "auto". This causes *Smrender* to try to find an angle in such a way that it does not colide (or at least as little as possible) possible which other objects that have already been rendered. In case of auto-rotation, the additional parameters weight, and phase may be set.
Smrender virtually rotates the caption from 0 to 360 degrees and calculates the color difference between the foreground (the caption) and the background for each angle. It then chooses the angle with the greatest color difference which should be the place where it is best visible. If the angle is between 90 and 270 degrees, *Smrender* automatically flips the caption that it does not read upside-down.
The parameters weight and phase may influence the auto-rotation if specified. The weight is a decimal value between 0 and 1 (1 is default) which allows to weight the angles of 90 plus phase and 270 plus phase less than the others (a phase of 0 is default). This allows to e.g. prefer left-right angles above top-bottom angles. This makes sense because reading left-right is more easy than reading top-bottom.
For example if "weight=0.7" is defined, northerly and southerly test samples are taken into account only with 70% which leads to the fact that the the caption tends to be rather east-west aligned.
The parameter 'anglekey' defines a key which can be used if the caption shall be rotated based on the value of a tag. The value of the parameter angle is added additionally. The parameter angle=auto is ignored if a anglekey is set.
If hide is set to yes or any other boolean true value (see Section 5.3) this function is executed as described above except that it finally does not display the caption. This may be useful if auto-rotation is used (angle = auto). In a first step cap() is called with hide = true. Subsequently other rules may be called by using e.g. the tags anglekey or alignkey.

Captions on ways are handled a little bit different from captions on nodes. Actually, captions on ways (polylines) are not supported yet but captions on areas (closed polygons) are supported very well, although it is still in development.
Smrender will calculate the centroid and the area of the polygon. The caption is than placed at the position of the centroid[9]. If the parameter size is omitted or set to 0, the font size is chosen dependent on the square root of the area. Thus, larger polygons get larger captions and smaller ones get smaller captions.

[7] `http://en.wikipedia.org/wiki/X11_color_names`
[8] Please note that this is different to the angle definition of maritime navigation which is degrees clockwise from 0 to 360 being 0 upwards (North).
[9] This is similar to what *Osmarender* does.

6.1.2. Drawing and Filling

The *draw* action is used to draw lines of various styles and fill polygons. This action uses solid colors for its operation. If you wish to fill an area with an image as pattern please have a look at Section 6.1.3. The following shows the basic rule format.[10]

```
<action>    := 'draw:' <param> [ ';' <param> ... ]
<param>     := <filldef> | <borderdef> | <options>
<filldef>   := <color> | <style> | <width> | <dashlen>
<borderdef> := <bcolor> | <bstyle> | <bwidth> | <bdashlen>
<color>     := 'color' '=' <coldef>
<bolor>     := 'bcolor' '=' <coldef>
<style>     := 'style' '=' <styledef>
<styledef>  := 'solid' | 'dashed' | 'dotted' | 'transparent' |
               'pipe' | 'rounddot'
<bstyle>    := 'bstyle' '=' <styledef>
<width>     := 'width' '=' <num>
<bwidth>    := 'bwidth' '=' <num>
<dashlen>   := 'dash' '=' <dashdef>
<dashdef>   := <length> [ ':' <length> ... ]
<options>   := <dirdef> | <opendef> | <curvedef> | <wavedef>
<dirdef>    := 'directional' '=' <bool>
<opendef>   := 'ignore_open' '=' <bool>
<curvedef>  := <curve> | <cfactor>
<curve>     := 'curve' '=' <bool>
<cfactor>   := 'curve_factor' '=' <decval>
<wavedef>   := <wave> | <wlen>
<wave>      := 'wavy' '=' <bool>
<wlen>      := 'wavy_length' '=' <bool>
```

The action behaves a little bit different if it is a polyline (open way, e.g. a river) or a polygon (closed way, i.e. an area, e.g. a lake).
Independently if it is an open or a closed polygon there is always a filled part which is enclosed with a border. The filled part is defined by color, width, and style at which just color is honored for closed polygons. The border part is defined by bcolor, bwidth, and bstyle.
The arguments to style and bstyle are one of dashed, dotted, solid, transparent, pipe, or rounddot. If a style parameter is omitted, a solid line is drawn.
Rounddot creates a dotted line like dottet but instead of short lines (squares) round dots are drawn.
Pipe is used to create a line style as is used on sea charts for underwater pipes. It is a combination of dotted and dashed.
Smrender allows to specify the length of dashes and the spaces in betweed with the options dash and bdash.
Width and bwidth are given in millimeters. A width of 0 draws the thinnest possible line.
If the boolean parameter curve is set to true, *Smrender* will use Bezier curves to smoothen lines and polygons. Optionally, curve_factor may be set to any decimal value which can be used to modify the curve extremities. A value of 0.0 disables the curving completely. The default value is defined by the macro DIV_PART which is currently set to 0.2.

[10] Basic BNF parameters are defined in Section 5.3 on page 10.

Smrender is able to generate wavy lines if the boolean parameter wavy is set. On sea charts this line style is used for underwater cables. Please note that *Smrender* internally uses Bezier curves to achieve this. Thus, wavy and curve are mutually exclusive. The wave length can be influenced with the decimal parameter wavy_length. Any known unit (see Section 5.4) may be used. Milimeters (mm) is the default if no unit is specified.
If ignore_open is set to "1", the rule is applied to closed polygons only.
A special fill mode is used if directional is set to "1". Usually, *Smrender* always fills the inner part of a polygon independently of its direction, i.e. if the nodes of the polygon (way) are ordered clockwise or counterclockwise. This mode is useful if areas of same type (same tags) are enclosed within each other. This may result in unexpected rendering results. The main reason for that is that OSM is just a two-dimensional database. OpenStreetmap provides special tagging facilities to handle such cases, for example multi-poly relations.
A typical application for the *directional* fill mode on sea charts is the rendering of depth contours, in particular if they are filled in shallow inshore areas. *Smrender* takes care on the direction of the polygons.[11] It always fills the portion which is left of the way. Figure 1 shows an example. Shallow water with a depth less than 10 meters is rendered blue, deeper areas are white (transparent). The white area northeast of the islet Radelj is such a 20 meters area which is enclosed by a more shallow area and than again by a deeper area on the west side of this chart detail.
Filling polygons using this mode works only if the polygons are edited correctly, i.e. their direction is correct. Furthermore it is slightly slower than the regular fill mode.

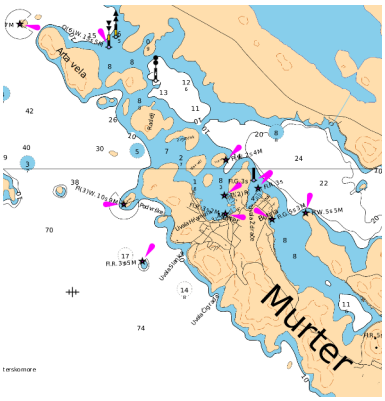

Figure 1: Directional filling.

6.1.3. Placing Images

Smrender allows to place images at the position of nodes or to fill areas using an image as pattern.

```
<definition>   := 'img:' [ <param> [ ';' <param> ... ]]
<param>        := <name> '=' <value>
<name>         := 'file' | 'angle' | 'scale' | 'mkarea' | 'anglekey'
```

The parameter file is mandatory and contains a path to a PNG file. If *Smrender* was compiled with librsvg2 (see Section A) it supports SVG files as well. The image is placed with its center directly at the position of the matching node without any modifications.
The parameter angle specifies an angle between 0 and 360 degrees to which to image may be rotated before it is placed onto the map.
If angle is set to "auto", *Smrender* tries to find a rotation angle. This works as described in Section 6.1.1. The rotation function does not take the image itself into account except its size. The rotation test starts at direction East and rotates counterclockwise. Obviously, this makes only sense if it is applied to asymmetric non-centered images, such as light flares. On areas, "auto" has no effect.
The paramter anglekey defines the name of the key of a tag which is used to derive the angle of rotation during runtime. It is implemented in the same way as it is for captions (see Section 6.1.1). Obviously, this does not work together with angle=auto. The latter is ignored if anglekey is set.

[11] A few features exist in OSM as well of which the rendering depends on the direction. Most importantly this is natural=coastline. Other examples are waterway=canal and natural=cliff.

The parameter scale allows to scale the image. A value greater than 1 will enlarge the image, if scale is less than 1 it will shrink the image. *Smrender* allows to set a global scaling parameter which is applied to all images, additionally. This is set with the command line parameter **-s** (see Section 4.2).
The parameter mkarea is a boolean parameter. If set to *yes*, the auto-rotation function will add nodes and ways to the data which indicate the level priority of the angles around the node. This is mainly intended for debugging.

6.2. Data Manipulation

6.2.1. Adding Objects

With this action you can add OSM objects as defined in the ruleset to the data. You can use this to add specific OSM objects during during the stage of rendering. Thus, you don't have to "pollute" your source data with map-specific data before. Typically, those objects added are graphically rendered by a subsequent rule. Currently, *Smrender* only OSM nodes are supported.

```
<definition>  := 'add:' [ <param> [ ';' <param> ... ]]
<param>       := <units> | <halign> | <valign> | <reference>
<units>       := 'units' '=' <decval> [ <udef> ]
<decval>      := a decimal number
<halign>      := 'halign' '=' <hdef>
<hdef>        := 'north' | 'south'
<valign>      := 'valign' '=' <vdef>
<vdef>        := 'east' | 'west'
<reference>   := 'reference' '=' <ref>
<ref>         := 'absolute' | 'relative'
```

This function adds a node to the OSM data. All tags of the node are copied except the _action_ tag. The position of the new node is derived from its latitude and longitude. By default, the coordinates are treated as absolute values referred to the geographic coordinate system. if reference is set to relative, the coordinates are treated as relative values to the center of the page. This can be influenced by the parameters halign and valign in which case the origin is set to a page border or page corner. The latitude and the longitude have to be set as decimal number. With the parameter units the unit of measurement can be changed. The default is degrees but mm (milimeters) or cm (centimeters) can be used as well. If relative referencing used, positiv values mean a shift to the right (eastern) and upwards (northern), negative values shift to the left (western) and downwards (southern).

Example The following example adds a node 7 centimeters to the right and above of the lower left page corner.

```
<node lat="70" lon="70">
  <tag k="compass" v="yes"/>
  <tag k="name" v="2°05'E 2003 (5'E)"/>
  <tag k="bearing" v="2.0833"/>
  <tag
    k="_action_"
    v="add:reference=relative;halign=west;valign=south;units=mm"
   />
</node>
```

6.2.2. Adding Tags to Objects

The action set_tags allows to add an arbitrary number of OSM tags to an object. The tags have to be defined through an object within the rules file. This object may have no action tag. The template should have an id because the action set_tags needs to have a reference to it. To format simple looks like the following.

```
<action>    := 'set_tags:' <param>
<param>     := <name> '=' <value>
<name>      := 'id'
```

Please note that the object type of the rule must be the same as the object type of the template.

6.2.3. Standard Shapes

Smrender is able to generate standard shapes like triangles, or circles using this action. The formal definition looks like the following.

```
<action>        := 'shape:' <param>
<param>         := <name> '=' <value>
<name>          := 'nodes' | 'style' | 'radius' |  'angle' |
                   'key' | 'weight' | 'phase' | 'start' |
                   'end' | 'subtype' | 'r2'
```

This action internally generates an ellipse[12] with the given radius (the semi-major axis a) and places a number of nodes on its circumference.
Because OSM does not support any kind of arcs natively, they are constructed using ways with a specific number of nodes. The parameter nodes specifies this number of nodes. Thus, for example, if nodes is set to 3, the result will be a triangle.
The parameter weight is set to 1 by default if it is omitted. It is a multiplier which is used to calculate the semi-minor axis $b = weight \times a$. Thus, if weight = 1 a circle is generated. The parameter phase shifts the points along the circumference in a counterclockwise order. Thus, the following action creates a rectangle of the dimension 4×1.2 millimeters which is rotated by 20 degrees counterclockwise.

```
<tag k='_action_'
     v='shape:nodes=4;radius=2;angle=20;weight=0.3;phase=45'/>
```

One of the parameters nodes or style is mandatory. The latter argument is a preset for a specific number of nodes. Currently the styles triangle (= 3 nodes), square (= 4 nodes), and circle (= maximum nodes) are supported.
The radius is given in millimeters and the optional parameter angle may be used to rotate the shape at any degrees counterclockwise. If radius is omitted 1 millimeter is used as a default value. With key the shape may be rotated dependent on the value of a tag of a node. Key defines the key of this tag.
This action does not render anything itself. It generates an according set of new nodes which are connected together with a way. All these nodes and the way get the tag generator=smrender. The way additionally inherits all tags of the original node which was matched by the rule set to invoke this action. These tags can then be used to render the shape with a way rule. See the following snippet as an example.

[12] Wikipedia: Ellipse, http://en.wikipedia.org/wiki/Ellipse.

```
<node version='-1'>
   <tag k='natural' v='peak'/>
   <tag k='_action_' v='shape:style=triangle;radius=.7'/>
</node>
<way>
   <tag k='natural' v='peak'/>
   <tag k='_action_' v='draw:color=#906030'/>
</way>
```

6.2.4. Create Formatted Strings

This action is intended to create formatted strings out of a set of tags. It works in a similar but yet more simple manner as printf(3) does. The newly constructed string will be added as a new tag to the OSM object. This tag may then be used to match on in subsequent rules.

```
<action>        := 'strfmt:' <param>
<param>         := <name> '=' <value>
<name>          := 'addtag' | 'format' | 'key'
```

Strfmt() has two mandatory arguments. The first one is addtag which contains the name for the new tag which will be added to this object. The second parameter is format which specifies a format string. It may contain any characters and a set of format symbols. The format symbols are the % character followed by one of the following characters.

- s The string of the value of the tag specified by the respective key is copied to the output string.
- f The string of the value of the tag specified by the respective key is interpreted as a floating point number and copied to the output string.
- d The string of the value of the tag specified by the respective key is interpreted as an integer number and copied to the output string.
- [n]r n digits of the fractional part of a floating pointer number is copied to the output string as integer number. The number n is optional. If it is omitted, 1 is assumed.

  Example: Let's assume that the original value is 3.1415, then the format specifier '%r' would produce '1' as result, and if '%2r' is specified the result would be '14'.
- % The percent character is literally copied to the output string.
- v A semicolon is copied to the output string.

All regular characters are directly copied to the output string without conversion. The action *must* contain a key for each format symbol in the format string. The keys are used exactly in the order as they appear in the action line of ruleset.

Format String Example The following example shows how to create a new string for all peaks. It shows the name of the peak and its elevation in parentheses. This string is then rendered in the second rule.

```
<node>
   <tag k='natural' v='peak'/>
   <tag k='_action_' v='strfmt:format=%s (%s);
                  addtag=peak_string;key=name;key=ele'/>
</node>
<node>
   <tag k='peak_string' v=''/>
   <tag k='_action_' v='cap:font=serif;size=2;key=peak_string'/>
</node>
```

6.2.5. Concatenating Ways

This function closes open polygons. To have closed polygons is highly important because just such polygons can be filled with a background color.
Polygons which are literally closed, such as the coastline or lakes are very often found as a set of open ways whose beginning and end share the same nodes. This is because different tags may be attached to different parts of the polygon. Furthermore, just partial data sets are used as input because typically just a small area out of the world's data is selected.

Cat_poly has three optional parameters: ign_incomplete, no_corner, and copy. The first two can both be set to either 0 or 1. If these parameters are omitted, both are internally set to 0 by default.
If ign_incomplete is set to 1, cat_poly will only close such polygons which are formed by a collection of ways where the end point of each way directly is the starting point of the next way.
If ign_incomplete is set to 0 (which is the default) cat_poly will also close polygons which are still open even if all ways which have direct neighbors are connected by inserting artificial ways. This is done by connecting the end of each way to the beginning of the next way in the clockwise order of the bearing from the center point of the image to the start/end nodes of the ways.
Please note that the insertion of artificial ways only works properly if the ways have a specific direction. This is true at least for the ways which are tagged with natural=coastline.
The parameter copy can occur multiple times and it is used to specify the keys of the tags of the ways which should be copied to the newly joined long way. If the values of these keys differ than *Smrender* takes just the first one. All others are ignored. This is because OSM defines that a tag can appear just excatly one time in an OSM object.

If cat_poly is applied in a way rule than *Smrender* tries to close all ways which match the criteria of the rule. The function finds all adjacent ways and closes them properly. The original data is not changed furthermore it creates and inserts new ways. Those new ways are tagged with all tags that where defined in the rule set plus the tag generator=smrender plus all tags which have been specified by the copy parameters.
As already mentioned, a typical application for this function is to close the coastline which will be open in most cases. The coastline is always tagged with natural=coastline.
If cat_poly is applied to a relation then *Smrender* closes all ways of each relation separately. It creates a new closed way which will receive all tags of the relation, respectively. Additionally, it adds the tag generator=smrender. The tags of the way segments are join to the new way if they are listed with the parameter copy as explained above.
Cat_poly applied to relations is useful for example in the Agean Sea, where all partial ways of an island are grouped together using relations. The tags of this relation contains global information

about each island, such as its name or population.

Please note that cat_poly() may create ways with more the 2000 nodes which violates the OSM standard definition.[13]. This may cause problems if an output file is created (e.g. with option **-w**) and used in other OSM applications. *Smrender* supports ways of up to 2^{31} nodes. It is assumed that most other OSM processing tools do not care about the artifical boundary of 2000.

6.2.6. Inherit Tags to Objects

```
<action>    := 'inherit_tags:' [ <param> [ ';' <param> ] ]
<param>     := <objdef> | <dirdef> | <force> | <keydef>
<objdef>    := 'object' '=' <obj>
<obj>       := 'node' | 'way' | 'relation'
<dirdef>    := 'direction' '=' <dir>
<dir>       := 'up' | 'down'
<force>     := 'force' '=' <bool>
<keydef>    := 'key' '=' <string>
<string>    := any valid osm key string
```

This action copies tags from objects to their *parent* or children objects depending on the value of direction. A relation is considered as the parent of all of its members and a way is the parent of all of its nodes.
There may be a list of one or more key parameters and optionally the parameters object and force. The keys specify which tags from the child (which is the object to which this rule is applied if direction=up) are copied to its parents – and vice versa if direction=down. If object is not defined, the tags are copied to all parents/children of any type. The parameter object may be set to either way or relation in which only the parents of those types are taken into consideration.
The parameter force may be set to 0 (which is default) or 1. In the latter case inherit_tags will overwrite tags in the parent if they do already exist.

6.2.7. Generating a Grid

```
grid()
```

This action creates a grid exactly like the command line option **-g** but it provides more options. It provides the parameters margin, tickswidth, and subtickswith to adjust the size of the axis rulers. Values are given in milimeters. Furthermore it provides the parameters grid, ticks, and subticks which work exactly like the command line option (see Section 4.2). Missing parameters are initialized with default values.

6.2.8. Create a Ruler

```
ruler()
```

This function allows to generate a metric ruler. It is thought to be used for land maps. It takes the arguments section and count. Section is the length of one section of the ruler in kilometers. Count sets the number of sections.

[13] See `http://wiki.openstreetmap.org/wiki/Way`.

6.2.9. Text Translation

```
<action>    := 'translate:' [ <param> [ ';' <param> ] ]
<param>     := <keydef> | <iddef> | <newtagdef>
<keydef>    := 'key' '=' <keyval>
<keyval>    := <string> | '/' <regex> '/'
<iddef>     := 'id' '=' <objid>
<newtagdef> := 'newtag' '=' <bool>
```

This function translates tag values. It can be used to e.g. translate text into different languages. The function looks up a given value in a translation table and replaces its value accordingly. If no entry is found in the translation table, no translation is performed.
The keys whose values should be translated are given with the parameter key. It can be specified several times to be applied to several tags of an object at once. The value of key can be a simple string or a regular expression enclosed in '/'s.
By default the values are replaced in-place. If the old value should be preserved, set the parameter newtag=1. *Smrender* will then insert a new tag into the list of tags. Its value will be the translated value and the name of the key will be appended by the string ':local'.

The translation table is defined in the ruleset as well like a template (see Section 6.2.2). This is a rule node with an action. It has to have a node id because it is referred to by the translate()-rule. This translation object (the template) lists key-value-pairs being the key k the original value and the value v the one translated one.
Have a look at Section 10.2 for a detailed example.

6.2.10. Special Data Manipulation Functions

- clip

  This function clips[14] to chart to a given border, i.e. it cuts of the rendered parts at the page border around to chart.

  The optional parameter to clip() is border which takes 4 decimal comma-delimited values. These are the 4 page border values (northern, eastern, southern, western) in millimeters, e.g. border=15,20,15,20. If border is omitted, *Smrender* clips to the default axis borders of the chart.

- dist_median (EXPERIMENTAL)

  This function calculates the median of the length of the edges of a way. It adds the tag smrender:dist_median=*. The value of the tag contains the result of the calculation.

- ins_eqdist (EXPERIMENTAL)

  This function inserts nodes along a way with equal distances. The distance is given with the parameter distance in any unit according to 5.4. If no unit is specified, nautical miles is taken as default. The newly inserted nodes inherit all tags from the way and additionally the tags generator=smrender, and distance=*, and bearing=* are added. The latter two tags are set to the appropriate values. The bearing may be used by the function shape (see Section 6.2.3) to create appropriate rotated shapes.

[14] https://en.wikipedia.org/wiki/Clipping_%28computer_graphics%29

- mask (EXPERIMENTAL)

  This function allows to mask nodes if there is a dense cluster of nodes in some places on the chart which would make the result to look like just as a blur of color. Sometimes this is also called *uncluttering*.

  The function has the parameter distance which gives the distance of nodes in nautical miles which should be taken into account. This means that if there are more than one node within the given distance, just the most centered one is "chosen". All other nodes are tagged with smrender:mask=yes. This tag can then be matched on within the ruleset.

- poly_area

  This function calculates the area of closed polygons in nautical square miles. It adds the tag smrender:area=* to the way. The value of the tag contains the area.

- poly_centroid

  This function calculates the centroid of a closed polygon. It then adds a new node at this position. The node will inherit all tags of the polygon. Additionally, the tag smrender:id:way=* is added whose value is set to the ID of the way, respectively.

- poly_len

  This function calculates the length of a polygon in nautical miles. It adds the tag smrender:length=* to the way.

- set_ccw and set_cw

  Those functions set the direction of a closed way to either clockwise ("cw") or counterclockwise ("ccw").

- split

  This action can be applied to nodes only. It splits all ways at the matching node into two parts. As a result two ways exist: the 1st one ends at the specific node and the 2nd one begins at the same node.

  The function split() has the options boolean argument remsegment. If it is set to true the way will be split at the matching node but the following segment will be removed. As a result there will also be two ways but they are seperated by one segment, the one which follows the matching node in the original way.

- refine_poly (DEPRECATED)

  This function smooths the edges of polygons. It may take the function arguments iteration and deviation. The first defines the number of loops of the iterative refinement process (default = 3). The latter defines the maximum deviation of the original polyline in meters (default = 50). This avoids too high distortion of the polylines.

- reverse_way

  This function always reverses a way independent of its current direction.

- transcoord

  This function takes the two arguments tlat and tlon which translates the point of reference. I.e. it rotates the surface of the world.

This is necessary to implement the Spilhaus projection which rotates the point of reference to 49.56371678 South and 66.94970198 East. This can be accomplished with the following rule:

```
<node version='-20' visible='true'>
   <tag
      k='_action_'
      v='transcoord:tlat=-49.56371678;tlon=66.94970198'
   />
</node>
```

- wrapdetect

  This function detects if a way wraps from East to West or from West to East across the 180th degree Meridian (West or East).

  If such a wrapping is detected it adds the tag smrender:wrapdetect=split to the node before the wrap. This could be used in a consecutive step to call the function split().

  This may be desireable because a lot of chart software (such as *Josm* and *Smrender* itself) cannot handle such situations properly.

  The ruleset could look like the following:

```
<way version='-15'>
   <tag k='_action_' v='wrapdetect'/>
</way>

<node version='-15' visible='true'>
   <tag k='smrender:wrapdetect' v='split'/>
   <tag k='_action_' v='split:remsegment=1'/>
</node>
```

- zeroway (EXPERIMENTAL)

  Insert an artificial way of length zero at the matching node if the node is connecting two other ways. Such a way is a way with two nodes where both nodes have the same position.

6.3. Special Purpose Functions

6.3.1. Calling External Functions

Smrender has the ability to call user-defined library functions. This feature provides modularity and the flexibility to be extended on the fly without modifying the core. Thus, *Smrender* can be used for nearly every kind of rule-based OSM file processing. The library calls dlopen(3) and dlsym(3) are used to dynamically import those functions.
The basic rule format is defined in the following.

```
<definition>  := <function> '@' <library> [':' <param>
                 [ ';' <param> ... ]]
<library>     := path/name of shared library
<param>       := <name> '=' <value>
```

Function is the name of the function as it is exported by the shared object library. In particular, the exported symbol has to be named act_function_main(), i.e. it has to be prefixed by "act_" and suffixed by "_main". If library contains a '/', the path is resolved and the shared object loaded from that location. Otherwise the dynamic linker tries to find the library in the appropriate system directories.[15]
Beside linking the function itself, *Smrender* tries to import the optional functions act_function_ini() and act_function_fini(). These two functions may be used for initialization and finalization of the main function.
The function is called on each match of an OSM node. The initialization function act_function_ini() is called once directly after the rules file was parsed before the first match. The finalization function act_function_fini() is called onced directly after the last match.
The prototypes are defined as follows.

```
int (*act_function_ini)(smrule_t*);
int (*act_function_main)(smrule_t*, osm_obj_t*);
int (*act_function_fini)(smrule_t*);
```

Act_function_main() gets a pointer to the rule structure and the OSM object which matched the rule. The object can be either a *node*, a *way*, or a *relation*.

```
typedef struct smrule smrule_t;
typedef struct action action_t;

struct smrule
{
    osm_obj_t *oo;
    void *data;         // arbitrary data
    action_t *act;
};

char *get_param(const char*, double*, const action_t*);
char *get_parami(const char*, int*, const action_t*);
```

The rule structure contains three pointers. The first one points to the OSM object of the rule as defined in the ruleset. The _action_ tag was removed by the rules parser. The Second pointer is initialized to NULL by *Smrender* and is not touched any further. It is thought to be used by the external functions to store arbitrary data. Please note that all resources that have been claimed by the _ini() function (such as heap memory) have to be freed again by the finalization function _fini(). The third pointer of type action_t contains all data which needs *Smrender* for rule processing. Its contents should not be touched except you know what you are doing. It is important that the action structure (action_t) contains the parameters which may have been passed to the function through the ruleset. The function get_param() shall be used to retrieve their values. The first parameter is a constant string to the name of the parameter. The second parameter is a pointer to a double variable which will receive the converted value of the parameter. Of course this works only if the parameter contains a decimal value. This pointer may be set to NULL if it is not used. The third parameter to get_param() is a pointer to the action structure of the rule.

```
typedef struct osm_obj
{
    // type of object: {OSM_NODE, OSM_WAY, OSM_REL}
    short type;
    // visibility: {0, 1}
    short vis;
    // OSM id
    int64_t id;
```

[15] See dlopen(3) for details.

```
    // version, changeset, user id
    int ver, cs, uid;
    // Unix timestamp
    time_t tim;
    // number of tags
    short tag_cnt;
    // Pointer to tags
    struct otag *otag;
} osm_obj_t;
```

The type of object can be determined on examination of osm_obj_t.type. The variable may be set to either of OSM_NODE, OSM_WAY, or OSM_REL. The object can then be type-casted to either a osm_node_t, a osm_way_t, or a osm_rel_t. All those OSM types are defined in osm_inplace.h.
The return value of the function controls the further behavior of *Smrender* while applying this same rule. A return value of 0 means no error. *Smrender* will call the function again at the next matching object. If the return value is greater than 0 it behaves similar but outputs a message in the log file. The message contains the return value. If a negative value is returned, *Smrender* immediately stops applying this rule, calls the _fini() function and processes the next rule.
Section B explains how to write own (rendering) functions more in detail.

Security Implications This feature basically allows any user to call arbitrary functions on the system. Thus, *Smrender* should **never ever** be installed with file modes SUID/GUID-root! This would be a potential security risk and might allow an attacker with access to your system to compromise it.

6.3.2. Output of OSM Data

With the action out it is possible to create an OSM file which contains all the objects which match. *Smrender* will create one file for each action. This means that if the same file name is used in several out actions, the latter will overwrite the earlier ones. The action takes just one argument, the path to the file.

```
<definition>  := 'out:' [ <param> [ ';' <param> ... ]]
<param>       := <name> '=' <value>
<name>        := 'file'
```

6.3.3. Executing Programs and Scripts

Smrender is able to run external 3rd-party programs and scripts. *Smrender* communicates through stdin, stdout, and stderr of the program. The new process is executed by the system call execvp(3) or execvpe(3) if available.

```
<action>      := 'exec:' <param>
<param>       := <name> '=' <value>
<name>        := 'cmd' | 'arg' | 'env' | 'osmhdr'
```

The mandatory parameter cmd defines the path to program. Optionally, one or more arguments can be passed to the program by multiply specifying arg. The arguments are passed exactly in the same order as in the action.
The optional parameter env can be used to set environmant variables. By default, the program is executed with an empty environment. *Smrender* provides an interactive interface to communicate with the process.

The parameter osmhdr is an optional boolean argument and influences the communication protocol as explained in the following Section.

The Communication Protocol is an interactive hybrid command line protocol. All information from *Smrender* to the process is sent in XML format. In turn for simplicity, the process can use simple commands.
The communication is initiated by *Smrender* with an XML header and the *Smrender* header. The latter contains the version of the protocol and the a string for the XML generator, similar to the OSM format.

```
<?xml version='1.0' encoding='UTF-8'?>
<smrender version='0.1' generator='smrender 3.0.r1535'>
```

After this header all OSM objects which machted the rule are sent, one after the other in OSM format version 0.6. If the action has the parameter osmhdr set, each object is included in an OSM header as well. The following shows an example of an object including the OSM header, i.e. osmhdr=yes.

```
<osm version='0.6' generator='smrender'>
<node id="39273652" version="5" timestamp="2009-02-05T06:45:28Z"
   uid="67265" visible="true" lat="44.2128480" lon="15.4482687">
<tag k="created_by" v="Merkaartor 0.13"/>
</node>
</osm>
```

If osmhdr is omitted or set to no, the first and the last line of the above stanza are suppressed. After each OSM object, *Smrender* waits for commands of the process. Every command will generate some output and finally send a status code. The status code may be used by the process determine if a command could be executed successfully.

```
<status code="200">OK</status>
```

The following commands are currently implemented:

- . [<code>]

  A single period on a line will provoke *Smrender* to send the next object which matches the rule. Optionally a numeric code between -128 and 127 can be supplied. 0 is default (if omitted) and simply means success. *Smrender* appends the status code 200 to each object. If no more objects are available, status code 404 is sent. In that case *Smrender* waits for a last confirmation through a single period.

  A positiv number indicates an error. *Smrender* will stop further processing of objects of this rule. The process has the chance to finalize and has to commit this with a single period again. *Smrender* will close all streams and continue executing the next rule.

  If A negative number is returned, *Smrender* interpretes this as a fatal error. As a consequence it will imediately stop further processing.

- get (node|way|relation) <id>

  Additional OSM objects can be retrieved with this command.

6.3.4. Special Control Functions

- diff (EXPERIMENTAL)

  This function takes the argument file (exactly like in the function out. See Section 6.3.2.) and infile. It compares the ids of all objects in infile with all objects wich have been loaded by *Smrender* (option **-i**) and writes all objects which do not exist to file.

- disable

  This function disables the object to which it is applied. This is done by setting the attribute visible="false". It is intended to be used to completely ignore certain objects during the rendering process. Please note that disabled objects cannot be re-enabled again because the rendering engine completely ignores invisible objects.

- enable_rule and disable_rule

  These actions take the mandatory argument id which specifies the id of the rule to be enabled or disabled. They actually set the visibility to either true or false (see Section 5). Please note that enabling a rule which would have been rendered before this one according to its version and id is not executed it again.

- exit

  This function forces *Smrender* to stop. No further rules will be processed and *Smrender* will create the output files according to the command line options. It behaves exactly like sending the INT signal (pressing ^C, see Section 7).

  This function is mainly intended for debugging rule sets.

- incomplete (EXPERIMENTAL)

  This action can be applied to relations only. It takes the single parameter file which specifies the name of the output file to which it will write the types and ids of all objects which are listed as members of the relation but are not found in the input data. This may be useful to download these objects to complete the input data if necessary.

  Please note that this currently may not work properly if you specify the same file name in several incomplete-rules.

- neighbortile (EXPERIMENTAL) This is a special function which may be handy in the tile creation process. It creates a directory named "neighbor_tiles" and within this a tile directory tree (zoom/x/y.conf). The files created within this will contain several variables suitable for being sourced in a shell script. The variables in the files specifies the bounding boxes of the tiles in zoom level 10. This function will be improved in future and the file format may change.

- sub (EXPERIMENTAL) Call a set of sub-rules.

7. Signals

Smrender installs two signal handlers, one for SIGUSR1 and one for SIGINT.
If *Smrender* receives a USR1 signal during the process of reading OSM input data, it outputs some statistics about the current position of reading and data throughput. It may be used as progress indicator if huge files are used as input. If *Smrender* receives a INT signal (which is typically generated by pressing ^C) during rendering, it immediately aborts rendering of further objects and

saves the image in its current state and exits normally. If SIGINT is caught twice, *Smrender* exits immediately.

8. Extensions

As explained in Section 6.3.1, *Smrender* is able to call functions of shared objects through dynamic linking at runtime. Thus it is very easy to extend the core functionality of *Smrender*. Currently, it comes with one additional library which is libsmfilter.

8.1. Libsmfilter and Smfilter

Smfilter[16] is a preprocessor for Osmarender. It adds some sea chart specific virtual nodes and ways to simplify the rendering process. The functionality of smfilter is now integrated into *Smrender* with libsmfilter. As a result, smfilter is no longer supported. Libsmfilter exports four functions: vsector(), pchar(), compass(), and sounding(). The first is the replacement for smfilter, the next is a new function which generates combined strings for light descriptions, the third creates a magnetic variation compass, and the last generates nodes and ways for depth soundings as used in sea charts.
All functions belong to the category *data manipulation* (see Section 6.2), which add OSM objects to the data. Thus, graphical rendering rules (see Section 6.1) are necessary to display those objects on the page.

8.1.1. Generating Light Sectors with vsector()

This function is a full replacement for smfilter. Smfilter took several options[17] to adjust the rendering behavior. These are the options **-a**, **-b**, **-d**, and **-r** in particular. Libsmfilter now takes exactly the same parameters since it is just a port. The parameters must be fed to it within the rules file. This was commonly described in Section 6.3.1. In detail the format looks like the following.

```
<param-str>    := <a-v-pair>[';' <a-v-pair>[';' ...]]
<a-v-pair>     := <attribute> '=' <value>
<attribute>    := 'a' | 'b' | 'd' | 'r' | 'f'
<value>        := decimal number
```

a This sets the maximum distance of arc nodes. Basically, the distance is scaled with the radius; the smaller the radius the smaller the distance and vice versa. With large radii the distance of nodes is limited to dist. The value is given in nautical miles.

b Leading and directional lights are rendered with a bearing line and a small arc at its end. deg sets the angle of this arc to one side which means it is drawn def degrees clockwise and def degress counterclockwise from the bearing line.

d This defines the arc divisor which is used to determine the distance of the arc nodes. Thus, the node distance equals the radius divided by div. (see also option -a).

r A light may not have any radius specified since the tag is optional. In such cases smfilter picks radius as default value.

[16] See `http://www.abenteuerland.at/smfilter/`
[17] See `http://www.abenteuerland.at/smfilter/smfilter.html`

f This is a scaling factor for all radii. In particular, this can be useful for charts in very large scales. The default value is 1.

This is an example for calling vsector() from the rule set.

```
<node>
   <tag k='seamark:type' v=''/>
   <tag
      k='_action_'
      v='vsector@libsmfilter.so:a=0.05;d=20;r=0.5'
   />
</node>
```

A full description of the output produced by vsector() is found in the smfilter(1) man page[18] and in the OSM wiki.[19]

8.1.2. Compatibility to Smfilter

The function vsector() does exactly the same as the original smfilter tool. Thus, they are considered to be nearly 100% compatible. The functionality is exactly the same but the file structure will still be different because *Smrender* processes the OSM file in a different way than smfilter.
The following shows two exchangable command lines, the first for smfilter and the second for *Smrender*.

```
smfilter -a 0.05 -d 20 -r 0.5 < in.osm > out.osm
```

```
smrender -i in.osm -o /dev/null -M -G -w out.osm
```

The following rules file has to be used in conjunction with *Smrender* to be a replacement for smfilter. Of course, the file may be extended with other rendering rules.

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version='0.6'>
   <node>
      <tag k='seamark:type' v=''/>
      <tag k='_action_'
           v='func:vsector@./libsmfilter.so?a=0.05,d=20,r=0.5'/>
   </node>
</osm>
```

8.1.3. Generating Light Description Strings

Pchar() generates a string which contains the characteristics of the light as it is used in official sea charts and the *List of Lights*. See Section P and P.16 in particular if the *Chart No. 1*.[20]
The function analyzes the tags of an object. If it contains valid *OpenSeamap* tags[21] it generates the light string and adds the new OSM tag seamark:ligh_character=* to the object. The value of the tag contains the string which may be rendered by a subsequent rule.

[18] http://www.abenteuerland.at/smfilter/smfilter.html
[19] http://wiki.openstreetmap.org/wiki/OpenSeaMap/smfilter
[20] http://www.nauticalcharts.noaa.gov/mcd/chart1/ChartNo1.pdf
[21] See http://wiki.openstreetmap.org/wiki/OpenSeaMap/Lights_Data_Model.

Since the description of lights varies in different languages, *Smrender* supports the optional parameter lang which may be set to either 'de' for German or 'hr' for Croatian language. If the parameter is omitted it defaults to English. The following table shows language examples according to *INT-1, IP 51*:

English	Fl.10s 40m 27M
German	Blz.10s 40m 27sm
Croation	B Bl 10s 40m 27M

8.1.4. Generating a Magnetic Variation Compass

This function creates a magnetic variation compass as usual on sea charts. This is a north-up compass with with crosslines rotated by the variation as shown in Fig. 2. As most other functions, it will not directly create graphics output but OSM nodes and ways with specific tags. Those can then be graphically rendered by using one of *Smrender*'s rendering primitives (s. Section 6.1.1 and 6.1.2).

```
<definition>   := 'compass@libsmfilter.so:' [ <param> [ ';'
                                               <param> ... ]]
<param>        := <name> '=' <value>
<name>         := 'variation' | 'radius' | 'ticks'
```

The parameter radius is mandatory and defines the outer radius of the compass. It is given as degrees on a Meridian.[22]
The parameter variation sets the variation, i.e. the amount of degress to which the crosslines will be rotated. It is set in degrees. A positive value denotes eastern variations.
The parameter ticks defines the number of ticks on the circle. The default value is 360.

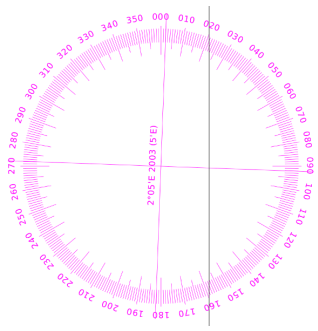

Figure 2: Magnetic variation.

All newly created nodes and ways are tagged with smrender:compass=* containig the bearing of the way in degrees. Every tenth outer node is additionally tagged with smrender:compass:description=* containing the bearing in degrees as well but with a leading zero as usual for the description of degrees on a sea chart.
See Section 10.1 for a complete example.

8.1.5. Generating Circles around Depth Soundings

Libsmfilter provides the function sounding() which generates small circles around depth soundings.
It generates symbols such as I.4 of Chart No. 1 and circles with a dashed line used for approximate depths (I.31).
Tags use for it is seamark:sounding=* containing the depth in meters, and optional seamark:sounding:quality = {approx | reported_unconfirmed}. See "Rendering Depths with *Smrender*"[23] for more details.

[22] This is subject to being improved in future releases.
[23] http://www.cypherpunk.at/2012/03/11/rendering-depths-with-smrender/

9. Usage Examples

There is a very simple example for your first rendered map. Before, download, compile, and install *Smrender* as explained in Section A.
Create a working directory somewhere. From the *Smrender* download URL[24] get the seamap icons package (icons.tbz2) and extract it.

```
tar xvfj icons.tbz2
```

Now we have to get some OSM data. We just use the Overpass API[25] to get a small window.

```
wget -O cr.osm \
   'http://www.overpass-api.de/api/xapi?map?bbox=15,43.7,15.4,44'
```

Now we can start *Smrender* by using the rule set rules.osm which comes with the *Smrender* package. Copy it to your working directory. By default it is installed into /usr/local/share/smrender.

9.1. Generating a PDF

PDF files are the best choice if you intend to print something. The following command line renders the OSM file cr.osm to the output image cr.png having the dimension of an A4 landscape page. 43° N 52.8' 15° E 12.8' are the center coordinates and the scale is chosen to be 1:100000 which is typical for sea charts.

```
smrender -i cr.osm -o cr.png -P A4 -l 43N52.8:15E12.8:100000
```

The result is the PNG image cr.png. You may look at it using your favorite image viewer. To get a correct non-distorted print-out it usually is a good idea to use the PDF format instead. If you try to print the image with a graphics program it most probably will be rescaled to fit the print margins. This will usually not happen if you print a PDF which has valid paper dimensions. To create a PDF file use the option **-O** instead.

```
smrender -i cr.osm -O cr.pdf -P A4 -l 43N52.8:15E12.8:100000
```

9.2. Generating a KAP File

KAP files are used by many applications that deal with marine navigation (e.g. *OpenCPN*, `http://opencpn.org/`) or GPS chart plotters or smart phones (e.g. *Marine Navigator* for Android, `https://play.google.com/store/apps/details?id=de.kemiro.marinenavigator`). Very often those files are also referred to as BSB files or also RNC files.
The following command generates a KAP file. It assumes that you have an input OSM file cr.osm. See the beginning of this Section (Section 9) on how to retrieve it. The density is reduced to 200 dpi (option **-d**) to save resources of your smart phone or chart plotter. Option **-G** disables the grid since most applications are able to generate a grid on their own. The KAP file is saved to cr.kap. The option **-s** 1 disables antialiasing. This also reduces resource usage because it limits the color space. Many plotting applications may have their built-in antialiasing.

```
smrender -i cr.osm -d 200 -G -k cr.kap -s 1 43N52.8:15E12.8:100000
```

[24] `http://www.abenteuerland.at/smrender/download/`
[25] `http://wiki.openstreetmap.org/wiki/Overpass_API`.

Please note that a PNG file and a KAP file may be generated at the same time by simply adding option **-o**. The file cr.kap can be used by your favorite application. If you use e.g. *Marine Navigator* on Android you have to copy the file to a folder named BSB_ROOT in the root directory of your SD card.

10. Ruleset Examples

10.1. Creating a Magnetic Variation Compass

```
<node lat="70" lon="70" version="-10">
   <tag k="compass" v="yes"/>
   <tag k="name" v="2°05'E 2003 (5'E)"/>
   <tag k="bearing" v="2.0833"/>
   <tag
    k="_action_"
    v="add:reference=relative;halign=west;valign=south;units=mm"
    />
</node>
<node version="-10">
   <tag k="compass" v="yes"/>
   <tag
    k="_action_"
    v="compass@libsmfilter.so:radius=0.03;variation=2.0833"
    />
</node>
<node>
   <tag k="compass" v="yes"/>
   <tag
    k='_action_'
    v='cap:font=sans-serif;color=magenta;size=2;key=name;
          anglekey=bearing;angle=-270;valign=north'
    />
</node>
<node>
   <tag k="smrender:compass:description" v=""/>
   <tag
    k='_action_'
    v='cap:font=sans-serif;color=magenta;size=2;
          key=smrender:compass:description;
          anglekey=smrender:compass;angle=0;valign=north'
    />
</node>
<way>
   <tag k="smrender:compass" v=""/>
   <tag k="_action_" v="draw:bcolor=magenta;width=0.2"/>
</way>
```

10.2. Translation Example

The following paragraphs demonstrate a translation of tags. First we need a translation table. The following shows a translation table for translating colors from English to German.

```
<node id='1000'>
   <tag k='red' v='rot'/>
```

```
   <tag k='green' v='grün'/>
   <tag k='blue' v='blau'/>
   <tag k='yellow' v='gelb'/>
</node>
```

Now we want to apply the translation to seamark nodes which have English names for the color of the lights. The name of these tags are either 'seamark:light:colour=*' or 'seamark:light:<N>:colour=*' where <N> is any integer greater than 0. A node can have multiple such tags. The rule should then look like this:

```
<node>
   <tag k='/seamark:light:.*colour/' v=''/>
   <tag
      k='_action_'
      v='translate:id=1000;key=/seamark:light:.*colour/;newtag=1'
   />
</node>
```

Let's look at the following node which could be within the input data.

```
<node id='12345678' version='1' lat='47.1234' lon='10.2323'>
   <tag k='name' v='Red-White-Lighthouse'/>
   <tag k='seamark:light:1:colour' v='yellow'/>
   <tag k='seamark:light:2:colour' v='red'/>
   <tag k='seamark:type' v='beacon_lateral'/>
</node>
```

Smrender will modify it according to the translation rule of above like follows.

```
<node id='12345678' version='1' lat='47.1234' lon='10.2323'>
   <tag k='name' v='Red-White-Lighthouse'/>
   <tag k='seamark:light:1:colour' v='yellow'/>
   <tag k='seamark:light:1:colour:local' v='gelb'/>
   <tag k='seamark:light:2:colour' v='red'/>
   <tag k='seamark:light:2:colour:local' v='rot'/>
   <tag k='seamark:type' v='beacon_lateral'/>
</node>
```

11. Files

The *Smrender* package contains all source C files and headers. A configure script is provided to create appropriate Makefiles and build *Smrender* (see Section A). It contains all sources for the smfilter library (see Section 8.1) in the directory libsmfilter/ and a skeleton libary in the directory libskel/ which may be used as a starting point for own functions (see Section B). Due to an internal code reorganization many general purpose functions are moved to the separate library libsmrender. All respective sources are found in the directory libsmrender/.
The package contains futhermore different rule sets which may also be used as a basis for own rule sets. The main ruleset is found in the directory rules_100000 which is actively maintained. Older files are rules.osm, rulesbig.osm, and rules_land.osm which are still provided with *Smrender*.

12. Bugs and Caveats

Smrender does not validate the well-formedness of the OSM files. Thus, you may get unexpected rendering results if the file format is incorrect.
For more information please look at the project homepage at `http://www.abenteuerland.at/smrender/`.

13. Author

Smrender is written by Bernhard R. Fischer, `mailto:bf@abenteuerland.at`. The idea of the project was born in summer of 2010. The actual development started in October of 2011.

14. Copyright

Copyright 2011-2015 Bernhard R. Fischer.

This file is part of *Smrender*.

Smrender is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3 of the License.

Smrender is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with *Smrender*. If not, see <`http://www.gnu.org/licenses/`>.

A. Compiling and Installing

Smrender should be simple to compile. It depends on libcairo[26] and optionally on fontconfig and librsvg2. Although they are not mandatory it is still suggested to use those packages. With fontconfig you can specify simple font faces such as e.g. sans-serif or DejaVuSans and the package will handle the path resolution to the necessary font files. You can still use fonts even without fontconfig but you then have to specify the full paths for the font files in the cap() rules (see Section 6.1.1) such as e.g. /usr/share/fonts/truetype/dejavu/DejaVuSans.ttf. The package librsvg2 is used to read SVG images in your ruleset, otherwise it supports just PNG images (see Section 6.1.3).
If none of those packages is not installed, *Smrender* will still compile and it can be used for OSM data processing but not for rendering charts (graphics output).
Download the most recent *Smrender* package from its main URL.[27] The packages includes the source code, some sea chart rulesets and a few ruleset examples, and a skelleton library (libskel) to show how to implement an own library with extension functions. It further includes libsmfilter for rendering special sea chart features (see Section 8.1), and a complete set of SVG icons for sea charts (buoys, beacons,...).

Extract the package with xzcat | tar xv smrender-4.0.r$xxxx$.xz. Change into the newly extracted directory. Then run the configure script ./configure, then build it with make. It should compile fine. Finally, there is the executable smrender and the libsmfilter.so. The latter is not mandatory for running *Smrender* since it may just be loaded dynamically by the rule set (see Section 8.1). Those files may be installed into the appropriate directories on your system with sudo make install.[28]
Smrender is known to compile with gcc 4.x on Debian Linux (Lenny, Squeeze, and Wheezy), FreeBSD version 8.x - 10.x, OpenBSD 5.x, and Mac OSX. It should compile on most Unixoid plattforms without further troubles, maybe even on Windows with Cygwin.

B. Writing Own Rendering Functions

The *Smrender* package includes a skeleton library in the directory libskel/. It implements the library constructor and destructor, and the actual rule function together with its initialization and de-initialization functions.
The directory contains also a Makefile which shows how to compile the library.

Smrender exports several functions which may be called by the library. The following list shows the most imported ones. The prototypes are defined in smrender.h, smlog.h, or osm_inplace.h.

```
// Use smrender's standard logging. This function is defined in 'smlog.h' and
// works similar to syslog(3).
void log_msg(int, const char*, ...);

// Get an OSM object (OSM_NODE, OSM_WAY, OSM_REL) with the specifed id. This
// function returns a pointer to either an osm_node_t or osm_way_t or_osm_rel_t
// structure on success, or NULL on error.
void *get_object(int, int64_t);
```

[26] http://www.cairographics.org/
[27] http://www.abenteuerland.at/download/smrender/
[28] Root privileges are required to install.

```
// Add an OSM object to the memory. The function returns 0 on success,
// otherwise -1 is returned. Preexisting objects with the same id are simply
// overwritten.
int put_object(osm_obj_t*);

// These functions return unique ids for nodes and ways.
int64_t unique_node_id(void);
int64_t unique_way_id(void);

// Initialize an OSM object. The number of tags (type short) and the number of
// node references (type int) must be supplied. Currently, the functions always
// return a valid pointer to an object. The objects returned are just partially
// initialized (see 'osm_func.c').
osm_node_t *malloc_node(short);
osm_way_t *malloc_way(short, int);
osm_rel_t *malloc_rel(short, short);
```

C. FAQ

This section covers some questions and answer which might arise.

C.1. Why is Smrender not written in C++?

On closer examination, the software architecture suggests an object-oriented programming language such as C++ but *Smrender* is written in C. The short answer is that C is always my first choice and the code was already too mature to switch to C++ without a high effort. The long answer is that *Smrender* is able to dynamically link libraries at runtime. Interfacing from C++ to a library written in C (currently) seems to be difficult (although not impossible).
But I still have in mind to rewrite *Smrender* in C++ when time comes.

C.2. Is it possible to create other maps, such as road maps?

Yes of course! The appearance of the map solely depends on the ruleset. A very simple first "land" ruleset is found in the rules directory. The only thing which is fixed is that *Smrender* uses a Mercartor projection which typically is not used for "land maps".

D. ToDo

Software is never finished and I have a long list of ideas in mind which I could implement.

- Rendering of rotated (not North-up) maps.
- Dynamic rules; these are rules which are generated during the rendering itself and are applied subsequently.
- Automatic font size for bays and capes (similar to the font size selection algorithm of polygons/islands).
- *Smrender* as an OSM database server with a Websockets interface (development started with version 4.0).