Package: rosmium (via r-universe)

May 14, 2026

Title Bindings for 'Osmium Tool'

Version 0.1.0.9000

Description Allows one to use 'Osmium Tool' (<https://osmcode.org/osmium-tool/>) from R. 'Osmium' is a multipurpose command line tool that enables one to manipulate and analyze OpenStreetMap files through several different commands. Currently, this package does not aim to offer functions that cover the entire 'Osmium' API, instead making available functions that wrap only a very limited set of its features.

License MIT + file LICENSE

URL https://ipeagit.github.io/rosmium/, https://github.com/ipeaGIT/rosmium

BugReports https://github.com/ipeaGIT/rosmium/issues

Imports checkmate, geojsonsf, processx, sf, utils

Suggests ggplot2, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://ipea.r-universe.dev

Date/Publication 2025-02-19 13:50:23 UTC

RemoteUrl https://github.com/ipea/rosmium

RemoteRef HEAD

RemoteSha 55318385656783c6997875f95da6e0a19b44803e

Contents

`extract` *Create geographical extracts from an OSM file*

Description

Creates geographical extracts from an OSM data file or an OSM history file. The geographical extent can be given either as a bounding box or as a (multi)polygon.

Usage

```
extract(
  input_path,
  extent,
  output_path,
  strategy = c("complete_ways", "smart", "simple"),
  overwrite = FALSE,
  echo_cmd = FALSE,
  echo = TRUE,
  spinner = TRUE,
  verbose = FALSE
)
```

Arguments

`input_path`	A string. The path to the OSM data/history file whose extent should be extracted from. Please see file_formats for a list of supported file formats.
`extent`	Either a `POLYGON` or a `MULTIPOLYGON` `sf` object with only one feature or a bbox object, created by sf::st_bbox.
`output_path`	A string. The path to the file where the output should be written to. Please see file_formats for a list of supported file formats.
`strategy`	A string. The strategy to use when creating the extract. Available strategies are `"complete_ways"`, `"smart"` and `"simple"`. Defaults to `"complete_ways"`. Please see the "Strategies" section for a description of each one of them.
`overwrite`	A logical. Whether existing files should be overwritten by the output. Defaults to `FALSE`.
`echo_cmd`	A logical. Whether to print the Osmium command generated by the function call to the screen. Defaults to `FALSE`.

echo	A logical. Whether to print the standard output and error generated by the Osmium call to the screen. Defaults to TRUE.
spinner	A logical. Whether to show a reassuring spinner while the Osmium call is being executed. Defaults to TRUE.
verbose	A logical. Whether to display detailed information on the running command. Defaults to FALSE.

Value

The normalized path to the output file.

Strategies

Different strategies can be used when creating extracts. Depending on the strategy, different objects will end up in the extracts. The strategies differ in how much memory they need and how often they need to read the input file. The choice of strategy depends on how you want to use the generated extracts and how much memory and time you have.

- "simple" - runs in a single pass. The extract will contain all nodes inside the region and all ways referencing those nodes, as well as all relations referencing any nodes or ways already included. Ways crossing the region boundary will not be reference-complete. Relations will not be reference-complete. This strategy is fast, because it reads the input only once, but the result is not enough for most use cases. This strategy will not work for history files.
- "complete_ways" - runs in two passes. The extract will contain all nodes inside the region and all ways referencing those nodes, as well as all nodes referenced by those ways. The extract will also contain all relations referenced by nodes inside the region or ways already included and, recursively, their parent relations. The ways are reference-complete, but the relations are not.
- "smart" - runs in three passes. The extract will contain all nodes inside the region and all ways referencing those nodes, as well as all nodes referenced by those ways. The extract will also contain all relations referenced by nodes inside the region or ways already included and, recursively, their parent relations. The extract will also contain all nodes and ways (and the nodes they reference) referenced by relations tagged "type=multipolygon" directly referencing any nodes in the region or ways referencing nodes in the region. The ways are reference-complete, and all multipolygon relations referencing nodes in the regions or ways that have nodes in the region are reference-complete. Other relations are not reference-complete.

Examples

```
pbf_path <- system.file("extdata/cur.osm.pbf", package = "rosmium")

file.size(pbf_path)

# buffering the pbf bounding box 4000 meters inward and using the result
# extent to extract the osm data inside it. transforming the crs because
# inward buffers only work with projected crs

lines <- sf::st_read(pbf_path, layer = "lines", quiet = TRUE)
bbox <- sf::st_bbox(lines)
```

```
bbox_polygon <- sf::st_as_sf(sf::st_as_sfc(bbox))
smaller_bbox_poly <- sf::st_buffer(
  sf::st_transform(bbox_polygon, 5880),
  -4000
)
smaller_bbox_poly <- sf::st_transform(smaller_bbox_poly, 4326)

output_path <- extract(
  pbf_path,
  smaller_bbox_poly,
  tempfile(fileext = ".osm.pbf")
)

file.size(output_path)
```

`file_formats` *Supported file formats*

Description

Please note that most of this documentation has been adapted from Osmium documentation. Please see https://docs.osmcode.org/osmium/latest/osmium-file-formats.html for the original documentation.

File types:

OSM uses three types of files for its main data:

- Data files: these are the most common files. They contain the OSM data from a specific point in time. At most one version of every object (node, way or relation) is contained in this file. Deleted objects are not in this file. The usual suffix used is `.osm`.
- History files: these files contain not only the current version of an object, but their history too. So for any object (node, way or relation) there can be zero or more versions in this file. Deleted objects can also be in this file. The usual suffix used is `.osm` or `.osh`. Because sometimes the same suffix is used as for normal data files (`.osm`) and because there is no clear indicator in the header, it is not always clear what type of file you have in front of you.
- Change files: sometimes called *diff files* or *replication diffs*, these files contain the changes between one state of the OSM database and another state. Change files can contains several versions of an object and also deleted objects. The usual suffix used is `.osc`.

All these files have in common that they contain OSM objects (nodes, ways and relations). History files and change files can contain several versions of the same object and also deleted objects; data files can't.

Where possible, Osmium commands can handle all file types. For some commands only some file types make sense.

Formats:

Osmium supports all major OSM file formats plus some more. These are:

- The classical XML format in the variants `.osm` (for data files), `.osh` (for data files with history) and `.osc` (for change files).
- The PBF binary format (usually with suffix `.osm.pbf` or just `.pbf`).
- The OPL format (usually with suffix `.osm.opl` or just `.opl`).
- The O5M/O5C format (usually with suffix `.o5m` or `.o5c`) (reading only).
- The "debug" format (usually with suffix `.osm.debug`) (writing only).

In addition, files in all formats except PBF can be compressed using *gzip* or *bzip2* (add `.gz` or `.bz2` suffixes, respectively - e.g. `data.osm.bz2`).

`show_content` *Show the content of an OSM file*

Description

Displays the content of an OSM file either in `.html`, `.xml` or `.opl` format.

Usage

```
show_content(
  input_path,
  output_format = c("html", "opl", "xml"),
  object_type = c("all", "node", "way", "relation", "changeset"),
  echo_cmd = FALSE,
  spinner = TRUE,
  preview = TRUE
)
```

Arguments

`input_path`	A string. The path to the OSM file whose content should be shown. Please see file_formats for a list of supported file formats.
`output_format`	A string. The format in which the content should be shown. Either `"html"`, `"opl"` or `"xml"`. Please note that the `"html"` option, although the most human readable format, results in large files that may be very slow to open and inspect, depending on the size of the input file.
`object_type`	A character vector. The object types that should be included in the output. Please note that using `"all"` is a simple way of passing all other available types (`"node"`, `"way"`, `"relation"` and `"changeset"`).
`echo_cmd`	A logical. Whether to print the Osmium command generated by the function call to the screen. Defaults to `FALSE`.
`spinner`	A logical. Whether to show a reassuring spinner while the Osmium call is being executed. Defaults to `TRUE`.
`preview`	A logical. Whether to open the temporary file in which the content of the OSM file was saved.

Value

The path to the temporary file in which the OSM file content was saved.

Examples

```
pbf_path <- system.file("extdata/cur.osm.pbf", package = "rosmium")

small_pbf <- tags_filter(
  pbf_path,
  "note",
  tempfile(fileext = "osm.pbf"),
  omit_referenced = TRUE
)

# both function calls below result in outputs with the same object types

show_content(small_pbf, object_type = "all")
show_content(
  small_pbf,
  object_type = c("node", "way", "relation", "changeset")
)

# only show nodes and ways

show_content(small_pbf, object_type = c("node", "way"))

# display all objects in xml format

show_content(small_pbf, output_format = "xml")
```

`tags_filter` *Filter objects matching specified keys/tags*

Description

Get objects matching at least one of the specified expressions from the input and write them to the output. All objects matching the expressions will be kept in the output, and objects referenced by them will also be added to the output (unless `omit_referenced = TRUE`). Objects will be written out in the order they are found in the input. Please note that the function will only work correctly on history files if `omit_referenced` is `TRUE`, and it cannot be used on change files.

Usage

```
tags_filter(
  input_path,
  filters,
  output_path,
  invert_match = FALSE,
```

```
  omit_referenced = FALSE,
  remove_tags = FALSE,
  overwrite = FALSE,
  echo_cmd = FALSE,
  echo = TRUE,
  spinner = TRUE,
  verbose = FALSE,
  progress = FALSE
)
```

Arguments

input_path	A string. The path to the OSM data/history file to which the filters should be applied. Please see file_formats for a list of supported file formats.
filters	A string. The filter expressions that should be applied to the input file. Please see the "Filter expressions" section for a description of the filter expression format.
output_path	A string. The path to the file where the output should be written to. Please see file_formats for a list of supported file formats.
invert_match	A logical. Whether to invert the sense of matching - i.e. to exclude objects with matching tags. Defaults to FALSE.
omit_referenced	A logical. Whether to omit the nodes referenced from matching ways and members referenced from matching relations. Defaults to FALSE.
remove_tags	A logical. Whether to remove tags from objects that are not matching the filter expression but are included to complete references (nodes in ways and members of relations). Defaults to FALSE. Please note that if an object is both matching the filter and used as a reference it will keep its tags.
overwrite	A logical. Whether existing files should be overwritten by the output. Defaults to FALSE.
echo_cmd	A logical. Whether to print the Osmium command generated by the function call to the screen. Defaults to FALSE.
echo	A logical. Whether to print the standard output and error generated by the Osmium call to the screen. Defaults to TRUE.
spinner	A logical. Whether to show a reassuring spinner while the Osmium call is being executed. Defaults to TRUE.
verbose	A logical. Whether to display detailed information on the running command. Defaults to FALSE.
progress	A logical. Whether to display a progress bar while running the command. Defaults to FALSE.

Value

The normalized path to the output file.

Filter expressions

A filter expression specifies a tag or tags that should be found in the data and the type of object (node, way or relation) that should be matched.

The object type(s) comes first, then a slash (/) and then the rest of the expression. Object types are specified as 'n' (for nodes), 'w' (for ways), 'r' (for relations) and 'a' (for areas - closed ways with 4 or more nodes and relations with `type=multipolygon` or `type=boundary` tag). Any combination of them can be used. If the object type is not specified, the expression matches all object types.

Some examples:

- `n/amenity` - matches all nodes with the key "amenity".
- `nw/highway` - matches all nodes or ways with the key "highway".
- `/note` - matches objects of any type with the key "note".
- `note` - matches objects of any type with the key "note".
- `w/highway=primary` - matches all ways with the key "highway" and value "primary".
- `w/highway!=primary` - matches all ways with the key "highway" and a value other than "primary".
- `r/type=multipolygon,boundary` - matches all relations with key "type" and value "multipolygon" or "boundary".
- `w/name,name:de=Kastanienallee,Kastanienstrasse` - matches any way with a "name" or "name:de" tag with the value "Kastanienallee" or "Kastanienstrasse".
- `n/addr:*` - matches all nodes with any key starting with "addr:"
- `n/name=*Paris` - matches all nodes with a name that contains the word "Paris".
- `a/building` - matches any closed ways with 4 or more nodes or relations tagged "building". Relations must also have a tag "type=multipolygon" or "type=boundary".

If there is no equal sign ("=") in the expression, only keys are matched and values can be anything. If there is an equal sign ("=") in the expression, the key is to the left and the value to the right. An exclamation sign ("!") before the equal sign means: a tag with that key, but not the value(s) to the right of the equal sign. A leading or trailing asterisk ("*") can be used for substring or prefix matching, respectively. Commas (",") can be used to separate several keys or values.

All filter expressions are case-sensitive. There is no way to escape the special characters such as "=", "*" and ",". You can not mix comma-expressions and "*"-expressions.

The specified filter expressions are matched in the order they are given. To achieve best performance, put expressions expected to match more often first.

Area matches (with leading "a/") do not check whether the matched object is a valid (multi)polygon, they only check whether an object might possibly be turned into a (multi)polygon. This is the case for all closed ways (where the first and last node are the same) with 4 or more nodes and for all relations that have an additional "type=multipolygon" or "type=boundary" tag.

Examples

```
pbf_path <- system.file("extdata/cur.osm.pbf", package = "rosmium")

# get all amenity nodes
output <- tags_filter(pbf_path, "n/amenity", tempfile(fileext = ".osm.pbf"))
```

```
nodes <- sf::st_read(output, layer = "points", quiet = TRUE)
head(nodes$other_tags)

# get all objects (nodes, ways or relations) with an addr:* tag
output <- tags_filter(
  pbf_path,
  "addr:*",
  tempfile(fileext = ".osm.pbf"),
  omit_referenced = TRUE
)
nodes <- sf::st_read(output, layer = "points", quiet = TRUE)
head(nodes$other_tags)

# get all nodes and ways with a highway tag and all relations tagged with
# type=restriction plus all referenced objects
output <- tags_filter(
  pbf_path,
  "nw/highway r/type=restriction",
  tempfile(fileext = ".osm.pbf")
)
ways <- sf::st_read(output, layer = "lines", quiet = TRUE)
head(ways$highway)
relations <- sf::st_read(output, layer = "other_relations", quiet = TRUE)
head(relations$other_tags)
```

`time_filter` *Create a time snapshot from an OSM history file*

Description

Filters objects from an OSM history file based on a specified point or interval in time. The output is a normal OSM data file containing the data as it would have looked at the specified time (UTC) or over the given interval.

Usage

```
time_filter(
  input_path,
  timestamp,
  output_path,
  overwrite = FALSE,
  echo_cmd = FALSE,
  echo = TRUE,
  spinner = TRUE,
  verbose = FALSE,
  progress = FALSE
)
```

Arguments

input_path	A string. The path to the OSM history file to be filtered. Please see file_formats for a list of supported file formats.
timestamp	A string, Date, or POSIXct, or a vector of two such values. The timestamp(s) in ISO 8601 format ('yyyy-mm-ddThh:mm:ssZ', e.g., "2015-01-01T00:00:00Z") or in ISO date format ('yyyy-mm-dd', e.g., "2015-01-01") representing the point in time for the snapshot or the start and end of a time interval. If Date or POSIXct objects are provided, they will be converted automatically to the required string format. **Note:** If two timestamps are provided and the first is later than the second, they will be swapped (with a warning) so that the interval is correct.
output_path	A string. The path to the file where the output should be written to. Please see file_formats for a list of supported file formats.
overwrite	A logical. Whether existing files should be overwritten by the output. Defaults to FALSE.
echo_cmd	A logical. Whether to print the Osmium command generated by the function call to the screen. Defaults to FALSE.
echo	A logical. Whether to print the standard output and error generated by the Osmium call to the screen. Defaults to TRUE.
spinner	A logical. Whether to show a reassuring spinner while the Osmium call is being executed. Defaults to TRUE.
verbose	A logical. Whether to display detailed information on the running command. Defaults to FALSE.
progress	A logical. Whether to display a progress bar while running the command. Defaults to FALSE.

Value

The normalized path to the output file.

Examples

```
history_path <- system.file("extdata/cur.osm.pbf", package = "rosmium")

# Single timestamp example (using ISO date format):
output_single <- time_filter(
  input_path = history_path,
  timestamp = "2015-01-01",
  output_path = tempfile(fileext = ".osm.pbf")
)

# Two timestamps example (mixing full ISO timestamp and ISO date format):
output_interval <- time_filter(
  input_path = history_path,
  timestamp = c("2015-01-01", "2015-06-01T00:00:00Z"),
  output_path = tempfile(fileext = ".osh.pbf")
)
```

Index