Package 'osmdata'

May 9, 2026

Title Import 'OpenStreetMap' Data as Simple Features or Spatial Objects

Version 0.3.0

Description Download and import of 'OpenStreetMap' ('OSM') data as 'sf' or 'sp' objects. 'OSM' data are extracted from the 'Overpass' web server (<https://overpass-api.de/>) and processed with very fast 'C++' routines for return to 'R'.

License GPL-3

URL https://docs.ropensci.org/osmdata/, https://github.com/ropensci/osmdata

BugReports https://github.com/ropensci/osmdata/issues

Depends R (>= 4.1)

Imports curl, httr2, methods, Rcpp (>= 0.12.4), rvest, tibble, utils, xml2

Suggests httptest2, jsonlite, knitr, markdown, rmarkdown, sf, sp, terra, testthat

LinkingTo Rcpp

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation yes

RoxygenNote 7.3.2

X-schema.org-applicationCategory Data Access

X-schema.org-isPartOf https://ropensci.org

X-schema.org-keywords open0street0map, openstreetmap, overpass0API, OSM

Author Joan Maspons [aut, cre] (ORCID: <https://orcid.org/0000-0003-2286-8727>), Mark Padgham [aut], Bob Rudis [aut], Robin Lovelace [aut],

Maëlle Salmon [aut],
Andrew Smith [ctb],
James Smith [ctb],
Andrea Gilardi [ctb],
Enrico Spinielli [ctb],
Anthony North [ctb],
Martin Machyna [ctb],
Marcin Kalicinski [ctb, cph] (Author of included RapidXML code),
Eli Pousson [ctb] (ORCID: <https://orcid.org/0000-0001-8280-1706>)

Maintainer Joan Maspons <joanmaspons@gmail.com>

Repository CRAN

Date/Publication 2025-08-23 23:30:02 UTC

Contents

`add_osm_feature` *Add a feature to an Overpass query*

Description

Add a feature to an Overpass query

Usage

```
add_osm_feature(
  opq,
  key,
  value,
  key_exact = TRUE,
  value_exact = TRUE,
  match_case = TRUE,
  bbox = NULL
)
```

Arguments

`opq`	An `overpass_query` object
`key`	feature key; can be negated with an initial exclamation mark, `key = "!this"`, and can also be a vector if `value` is missing.
`value`	value for feature key; can be negated with an initial exclamation mark, `value = "!this"`, and can also be a vector, `value = c("this", "that")`.
`key_exact`	If FALSE, `key` is not interpreted exactly; see https://wiki.openstreetmap.org/wiki/Overpass_API
`value_exact`	If FALSE, `value` is not interpreted exactly
`match_case`	If FALSE, matching for both `key` and `value` is not sensitive to case
`bbox`	optional bounding box for the feature query; must be set if no opq query bbox has been set

Value

opq object

`add_osm_feature` vs `add_osm_features`

Features defined within an add_osm_features call are combined with a logical OR.

Chained calls to either add_osm_feature or `add_osm_features()` combines features from these calls in a logical AND; this is analagous to chaining `dplyr::filter()` on a data frame.

`add_osm_features()` with only one feature is logically equivalent to `add_osm_feature()`.

Note

key_exact should generally be TRUE, because OSM uses a reasonably well defined set of possible keys, as returned by available_features. Setting key_exact = FALSE allows matching of regular expressions on OSM keys, as described in Section 6.1.5 of https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL. The actual query submitted to the overpass API can be obtained from opq_string.

References

https://wiki.openstreetmap.org/wiki/Map_Features

See Also

add_osm_features

Other queries: add_osm_features(), bbox_to_string(), getbb(), opq(), opq_around(), opq_csv(), opq_enclosing(), opq_osm_id(), opq_string(), overpass_status()

Examples

```
## Not run:
q <- opq ("portsmouth usa") |>
    add_osm_feature (
        key = "amenity",
        value = "restaurant"
    ) |>
    add_osm_feature (key = "amenity", value = "pub")
osmdata_sf (q) # all objects that are restaurants AND pubs (there are none!)
q1 <- opq ("portsmouth usa") |>
    add_osm_feature (
        key = "amenity",
        value = "restaurant"
    )
q2 <- opq ("portsmouth usa") |>
    add_osm_feature (key = "amenity", value = "pub")
c (osmdata_sf (q1), osmdata_sf (q2)) # all restaurants OR pubs

# Use of negation to extract all non-primary highways
q <- opq ("portsmouth uk") |>
    add_osm_feature (key = "highway", value = "!primary")

# key negation without warnings
q3 <- opq ("Vinçà", osm_type = "node") |>
    add_osm_feature (key = c ("name", "!name:ca"))
cat (opq_string (q3))
q4 <- opq ("el Carxe", osm_type = "node") |>
    add_osm_feature (key = "natural", value = "peak") |>
    add_osm_feature (key = "!ele")
cat (opq_string (q4))

# Get objects with keys (`natural` OR `waterway`) AND `name`
q_keys <- opq ("Badia del Vallès", osm_types = "nwr", out = "tags") |>
```

```
    add_osm_features (features = list (natural = NULL, waterway = NULL)) |>
    add_osm_feature (key = "name")
cat (opq_string (q_keys))

## End(Not run)
```

`add_osm_features` *Add multiple features to an Overpass query*

Description

Alternative version of add_osm_feature for creating single queries with multiple features. Key-value matching may be controlled by using the filter symbols described in https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL#By_tag_.28has-kv.29.

Usage

```
add_osm_features(
  opq,
  features,
  bbox = NULL,
  key_exact = TRUE,
  value_exact = TRUE
)
```

Arguments

opq	An overpass_query object
features	A named list or vector with the format list("<key>" = "<value>") or c("<key>" = "<value>") or a character vector of key-value pairs with keys and values enclosed in escape-formatted quotations. See examples for details.
bbox	optional bounding box for the feature query; must be set if no opq query bbox has been set.
key_exact	If FALSE, key is not interpreted exactly; see https://wiki.openstreetmap.org/wiki/Overpass_API
value_exact	If FALSE, value is not interpreted exactly

Value

opq object

add_osm_feature vs add_osm_features

Features defined within an add_osm_features call are combined with a logical OR.

Chained calls to either add_osm_feature or add_osm_features() combines features from these calls in a logical AND; this is analagous to chaining dplyr::filter() on a data frame.

add_osm_features() with only one feature is logically equivalent to add_osm_feature().

References

https://wiki.openstreetmap.org/wiki/Map_Features

See Also

add_osm_feature

Other queries: add_osm_feature(), bbox_to_string(), getbb(), opq(), opq_around(), opq_csv(), opq_enclosing(), opq_osm_id(), opq_string(), overpass_status()

Examples

```
## Not run:
q <- opq ("portsmouth usa") |>
    add_osm_features (features = list (
        "amenity" = "restaurant",
        "amenity" = "pub"
    ))

q <- opq ("portsmouth usa") |>
    add_osm_features (features = c (
        "\"amenity\"=\"restaurant\"",
        "\"amenity\"=\"pub\""
    ))
cat (opq_string (q))
# This extracts in a single query the same result as the following:
q1 <- opq ("portsmouth usa") |>
    add_osm_feature (
        key = "amenity",
        value = "restaurant"
    )
q2 <- opq ("portsmouth usa") |>
    add_osm_feature (key = "amenity", value = "pub")
c (osmdata_sf (q1), osmdata_sf (q2)) # all restaurants OR pubs

# Get objects with keys (`natural` OR `waterway`) AND `name`
q_keys <- opq ("Badia del Vallès", osm_types = "nwr", out = "tags") |>
    add_osm_features (features = list (natural = NULL, waterway = NULL)) |>
    add_osm_feature (key = "name")
cat (opq_string (q_keys))

## End(Not run)
```

---

available_features *List recognized features in OSM*

Description

List recognized features in OSM

Usage

```
available_features()
```

Value

character vector of all known features

Note

requires internet access

References

https://wiki.openstreetmap.org/wiki/Map_Features

See Also

Other osminfo: available_tags()

Examples

```
## Not run:
available_features ()

## End(Not run)
```

---

available_tags *List tags associated with a feature*

Description

List tags associated with a feature

Usage

```
available_tags(feature)
```

Arguments

feature	feature to retrieve

Value

character vector of all known tags for a feature

Note

requires internet access

References

https://wiki.openstreetmap.org/wiki/Map_Features

See Also

Other osminfo: available_features()

Examples

```
## Not run:
available_tags ("aerialway")

## End(Not run)
```

---

bbox_to_string *Convert a named matrix or a named or unnamed vector or data.frame to a string*

Description

This function converts a bounding box into a string for use in web apis

Usage

```
bbox_to_string(bbox)
```

Arguments

bbox	bounding box as character, matrix, vector or a data.frame with osm_type and osm_id columns. If character, the bbox will be found (geocoded) and extracted with getbb. Unnamed vectors will be sorted appropriately and must merely be in the order (x, y, x, y).

Value

A character string representing min x, min y, max x, and max y bounds. For example: "15.3152361,76.4406446,15.355236 is the bounding box for Hampi, India. For data.frames with OSM objects, a character string representing a set of OSM objects in overpass query language. For example: "relation(id:11747082)" represents the area of the Catalan Countries. A set of objects can also be represented for multirow data.frames (e.g. "relation(id:11747082,307833); way(id:22422490)").

See Also

Other queries: add_osm_feature(), add_osm_features(), getbb(), opq(), opq_around(), opq_csv(), opq_enclosing(), opq_osm_id(), opq_string(), overpass_status()

Examples

```
# Note input is (lon, lat) = (x, y) but output as needed for 'Overpass' API
# is (lat, lon) = (y, x).
bb <- c (-0.4325512, 39.2784496, -0.2725205, 39.566609)
bbox_to_string (bb)
# This is equivalent to:
## Not run:
bbox_to_string (getbb ("València"))
bbox_to_string (getbb ("València", format_out = "data.frame"))

## End(Not run)
```

`getbb` *Get bounding box for a given place name*

Description

This function uses the free Nominatim API provided by OpenStreetMap to find the bounding box (bb) associated with place names.

Usage

```
getbb(
  place_name,
  display_name_contains = NULL,
  viewbox = NULL,
  format_out = c("matrix", "data.frame", "string", "polygon", "sf_polygon",
    "osm_type_id"),
  base_url = "https://nominatim.openstreetmap.org",
  featuretype = "settlement",
  limit = 10,
  key = NULL,
  silent = TRUE
)
```

Arguments

`place_name`	The name of the place you're searching for
`display_name_contains`	Text string to match with display_name field returned by https://wiki.openstreetmap.org/wiki/Nominatim
`viewbox`	The bounds in which you're searching
`format_out`	Character string indicating output format: `matrix` (default), `string` (see `bbox_to_string()`), `data.frame` (all 'hits' returned by Nominatim), `sf_polygon` (for polygons that work with the sf package), `polygon` (full polygonal bounding boxes for each match) or `osm_type_id` (string for quering inside deffined OSM areas `bbox_to_string()`).

base_url	Base website from where data is queried
featuretype	The type of OSM feature (settlement is default; see Note)
limit	How many results should the API return?
key	The API key to use for services that require it
silent	Should the API be printed to screen? TRUE by default

Details

It was inspired by the functions bbox from the **sp** package, bb from the **tmaptools** package and bb_lookup from the github package **nominatim** package, which can be found at https://github.com/hrbrmstr/nominatim.

See https://wiki.openstreetmap.org/wiki/Nominatim for details.

Value

For format_out = "matrix", the default, return the bounding box:

```
  min   max
x ...   ...
y ...   ...
```

If format_out = "polygon", a list of polygons and multipolygons with one item for each nominatim result. The items are named with the OSM type and id. Each polygon is formed by one or more two-columns matrices of polygonal longitude-latitude points. The first matrix represents the outer boundary and the next ones represent holes. See examples below for illustration.

If format_out = "sf_polygon", a sf object. Each row correspond to a place_name within nominatim result.

For format_out = "osm_type_id", a character string representing an OSM object in overpass query language. For example: "relation(id:11747082)" represents the area of the Catalan Countries. If one exact match exists with potentially multiple polygonal boundaries, only the first relation or way is returned. A set of objects can also be represented for multiple results (e.g. relation(id:11747082,307833); way(id:22422490)). See examples below for illustration. The OSM objects that can be used as areas in overpass queries *must be closed rings* (ways or relations).

Note

Specific values of featuretype include "street", "city", https://wiki.openstreetmap.org/wiki/Nominatim for details). The default featuretype = "settlement" combines results from all intermediate levels below "country" and above "streets". If the bounding box or polygon of a city is desired, better results will usually be obtained with featuretype = "city".

See Also

Other queries: add_osm_feature(), add_osm_features(), bbox_to_string(), opq(), opq_around(), opq_csv(), opq_enclosing(), opq_osm_id(), opq_string(), overpass_status()

Examples

```
## Not run:
getbb ("Salzburg")
# select based on display_name, print query url
getbb ("Hereford", display_name_contains = "United States", silent = FALSE)
# top 3 matches as data frame
getbb ("Hereford", format_out = "data.frame", limit = 3)

# Examples of polygonal boundaries
bb <- getbb ("Milano, Italy", format_out = "polygon")
# A polygon and a multipolygon:
str (bb) # matrices of longitude/latitude pairs

bb_sf <- getbb ("kathmandu", format_out = "sf_polygon")
bb_sf
# sf:::plot.sf(bb_sf) # can be plotted if sf is installed
getbb ("london", format_out = "sf_polygon")

getbb ("València", format_out = "osm_type_id")
# select multiple areas with format_out = "osm_type_id"
areas <- getbb ("València", format_out = "data.frame")
bbox_to_string (areas [areas$osm_type != "node", ])

# Using an alternative service (locationiq requires an API key)
# add LOCATIONIQ=type_your_api_key_here to .Renviron:
key <- Sys.getenv ("LOCATIONIQ")
if (nchar (key) == 32) {
    getbb (place_name,
        base_url = "https://locationiq.org/v1/search.php",
        key = key
    )
}

## End(Not run)
```

`get_overpass_url` *get_overpass_url*

Description

Return the URL of the specified overpass API. Default is https://overpass-api.de/api/interpreter/.

Usage

```
get_overpass_url()
```

Value

The overpass API URL

See Also

set_overpass_url()

Other overpass: set_overpass_url()

Examples

```
get_overpass_url ()
```

---

opq *Build an Overpass query*

Description

Build an Overpass query

Usage

```
opq(
  bbox = NULL,
  nodes_only,
  osm_types = c("node", "way", "relation"),
  out = c("body", "tags", "meta", "skel", "tags center", "ids"),
  datetime = NULL,
  datetime2 = NULL,
  adiff = FALSE,
  timeout = 25,
  memsize
)
```

Arguments

bbox	Either (i) four numeric values specifying the maximal and minimal longitudes and latitudes, in the form c(xmin, ymin, xmax, ymax) or (ii) a character string in the form xmin,ymin,xmax,ymax. These will be passed to getbb to be converted to a numerical bounding box. Can also be (iii) a matrix representing a bounding polygon as returned from getbb(..., format_out = "polygon"). To search in an area, (iv) a character string with a relation or a (closed) way id in the format "way(id:1)", "relation(id:1, 2)" or "relation(id:1, 2, 3); way(id:2)" as returned by getbb(..., format_out = "osm_type_id") or bbox_to_string with a data.frame from getbb(..., format_out = "data.frame") to select all areas combined (relations and ways).
nodes_only	WARNING: this parameter is equivalent to osm_types = "node" and is DEPRECATED. If TRUE, query OSM nodes only.

osm_types	A character vector with several OSM types to query: node, way and relation is the default. nwr, nw, wr, nr and rel are also valid types. Some OSM structures such as place = "city" or highway = "traffic_signals" are represented by nodes only. Queries are built by default to return all nodes, ways, and relation, but this can be very inefficient for node-only queries. Setting this value to "node" for such cases makes queries more efficient and, in osmdata_sf(), the data will be returned in the osm_points list item only.
out	The level of verbosity of the overpass result: body (geometries and tags, the default), tags (tags without geometry), meta (like body + Timestamp, Version, Changeset, User, User ID of the last edition), skel (geometries only), tags center (tags without geometry + the coordinates of the center of the bounding box) and ids (type and id of the objects only).
datetime	If specified, a date and time to extract data from the OSM database as it was up to the specified date and time, as described at https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL#date. This *must* be in ISO8601 format ("YYYY-MM-DDThh:mm:ssZ"), where both the "T" and "Z" characters must be present.
datetime2	If specified, return the *difference* in the OSM database between datetime and datetime2, where datetime2 > datetime. See https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL#Difference_between_two_dates_(diff).
adiff	If TRUE, query for augmented difference. The result indicates what happened to the modified and deleted OSM objects. Requires datetime(2)*.
timeout	It may be necessary to increase this value for large queries, because the server may time out before all data are delivered.
memsize	The default memory size for the 'overpass' server in *bytes*; may need to be increased in order to handle large queries.

Details

The out statement for tags, tags centerand id, do not return geometries. Neither out = "meta" nor adiff = TRUE options are implemented for all osmdata_* functions yet. Use osmdata_xml or osmdata_data_frame to get the result of these queries. See the documentation of the out statement and augmented difference for more details about these options.

Value

An overpass_query object

Note

See https://wiki.openstreetmap.org/wiki/Overpass_API#Resource_management_options_.28osm-script.29 for explanation of timeout and memsize (or maxsize in overpass terms). Note in particular the comment that queries with arbitrarily large memsize are likely to be rejected.

See Also

Other queries: add_osm_feature(), add_osm_features(), bbox_to_string(), getbb(), opq_around(), opq_csv(), opq_enclosing(), opq_osm_id(), opq_string(), overpass_status()

Examples

```
## Not run:
q <- getbb ("portsmouth", display_name_contains = "United States") |>
    opq () |>
    add_osm_feature ("amenity", "restaurant") |>
    add_osm_feature ("amenity", "pub")
osmdata_sf (q) # all objects that are restaurants AND pubs (there are none!)
q1 <- getbb ("portsmouth", display_name_contains = "United States") |>
    opq () |>
    add_osm_feature ("amenity", "restaurant")
q2 <- getbb ("portsmouth", display_name_contains = "United States") |>
    opq () |>
    add_osm_feature ("amenity", "pub")
c (osmdata_sf (q1), osmdata_sf (q2)) # all restaurants OR pubs

# Use `osm_types = "node"` to retrieve single point data only, such as for central
# locations of cities.
opq <- opq (bbox, osm_types = "node") |>
    add_osm_feature (key = "place", value = "city") |>
    osmdata_sf (quiet = FALSE)

# Filter by a search area
qa1 <- getbb ("Catalan Countries", format_out = "osm_type_id") |>
    opq (osm_types = "node") |>
    add_osm_feature (key = "capital", value = "4")
opqa1 <- osmdata_sf (qa1)
# Filter by a multiple search areas
bb <- getbb ("Vilafranca", format_out = "data.frame")
qa2 <- bbox_to_string (bb [bb$osm_type != "node", ]) |>
    opq (osm_types = "node") |>
    add_osm_feature (key = "place")
opqa2 <- osmdata_sf (qa2)

## End(Not run)
```

---

opq_around *opq_around*

Description

Find all features around a given point, and optionally match specific 'key'-'value' pairs. This function is *not* intended to be combined with add_osm_feature, rather is only to be used in the sequence opq_around -> osmdata_xml (or other extraction function). See examples for how to use.

Usage

```
opq_around(lon, lat, radius = 15, key = NULL, value = NULL, timeout = 25)
```

Arguments

lon	Longitude of desired point
lat	Latitude of desired point
radius	Radius in metres around the point for which data should be extracted. Queries with large values for this parameter may fail.
key	(Optional) OSM key of enclosing data
value	(Optional) OSM value matching 'key' of enclosing data
timeout	It may be necessary to increase this value for large queries, because the server may time out before all data are delivered.

See Also

Other queries: add_osm_feature(), add_osm_features(), bbox_to_string(), getbb(), opq(), opq_csv(), opq_enclosing(), opq_osm_id(), opq_string(), overpass_status()

Examples

```
## Not run:
# Get all benches ("amenity=bench") within 100m of a particular point
lat <- 53.94542
lon <- -2.52017
key <- "amenity"
value <- "bench"
radius <- 100
x <- opq_around (lon, lat, radius, key, value) |>
    osmdata_sf ()

## End(Not run)
```

---

opq_csv *Transform an Overpass query to return the result in a csv format*

Description

Transform an Overpass query to return the result in a csv format

Usage

```
opq_csv(q, fields, header = TRUE)
```

Arguments

q	A opq string or an object of class overpass_query constructed with opq or alternative opq builders (+ add_osm_feature/s).
fields	a character vector with the field names.
header	if FALSE, do not ask for column names.

Details

The output format `csv`, ask for results in csv. See CSV output mode for details. To get the data, use osmdata_data_frame.

Value

The `overpass_query` or string with the prefix changed to return a csv.

Note

csv queries that reach the timeout will return a 0 row data.frame without any warning. Increase `timeout` in `q` if you don't see the expected result.

See Also

Other queries: `add_osm_feature()`, `add_osm_features()`, `bbox_to_string()`, `getbb()`, `opq()`, `opq_around()`, `opq_enclosing()`, `opq_osm_id()`, `opq_string()`, `overpass_status()`

Examples

```
## Not run:
q <- getbb ("Catalan Countries", format_out = "osm_type_id") |>
    opq (out = "tags center", osm_type = "relation", timeout = 100) |>
    add_osm_feature ("admin_level", "7") |>
    add_osm_feature ("boundary", "administrative") |>
    opq_csv (fields = c ("name", "::type", "::id", "::lat", "::lon"))
comarques <- osmdata_data_frame (q) # without timeout parameter, 0 rows

qid <- opq_osm_id (
    type = "relation",
    id = c ("341530", "1809102", "1664395", "343124"),
    out = "tags"
) |>
    opq_csv (fields = c ("name", "name:ca"))
cities <- osmdata_data_frame (qid)

## End(Not run)
```

---

opq_enclosing *opq_enclosing*

Description

Find all features which enclose a given point, and optionally match specific 'key'-'value' pairs. This function is *not* intended to be combined with add_osm_feature, rather is only to be used in the sequence opq_enclosing -> opq_string -> osmdata_xml (or other extraction function). See examples for how to use.

Usage

```
opq_enclosing(
  lon = NULL,
  lat = NULL,
  key = NULL,
  value = NULL,
  enclosing = "relation",
  timeout = 25
)
```

Arguments

lon	Longitude of desired point
lat	Latitude of desired point
key	(Optional) OSM key of enclosing data
value	(Optional) OSM value matching 'key' of enclosing data
enclosing	Either 'relation' or 'way' for whether to return enclosing objects of those respective types (where generally 'relation' will correspond to multipolygon objects, and 'way' to polygon objects).
timeout	It may be necessary to increase this value for large queries, because the server may time out before all data are delivered.

See Also

Other queries: add_osm_feature(), add_osm_features(), bbox_to_string(), getbb(), opq(), opq_around(), opq_csv(), opq_osm_id(), opq_string(), overpass_status()

Examples

```
## Not run:
# Get water body surrounding a particular point:
lat <- 54.33601
lon <- -3.07677
key <- "natural"
value <- "water"
x <- opq_enclosing (lon, lat, key, value) |>
    opq_string () |>
    osmdata_sf ()

## End(Not run)
```

opq_osm_id *Add a feature specified by OSM ID to an Overpass query*

Description

Add a feature specified by OSM ID to an Overpass query

Usage

```
opq_osm_id(
  id = NULL,
  type = NULL,
  open_url = FALSE,
  out = "body",
  datetime = NULL,
  datetime2 = NULL,
  adiff = FALSE,
  timeout = 25,
  memsize
)
```

Arguments

id	One or more official OSM identifiers (long-form integers), which must be entered as either a character or *numeric* value (because R does not support long-form integers). id can also be a character string prefixed with the id type, e.g. "relation/11158003"
type	Type of objects (recycled); must be either node, way, or relation. Optional if id is prefixed with the type.
open_url	If TRUE, open the OSM page of the specified object in web browser. Multiple objects (id values) will be opened in multiple pages.
out	The level of verbosity of the overpass result: body (geometries and tags, the default), tags (tags without geometry), meta (like body + Timestamp, Version, Changeset, User, User ID of the last edition), skel (geometries only), tags center (tags without geometry + the coordinates of the center of the bounding box) and ids (type and id of the objects only).
datetime	If specified, a date and time to extract data from the OSM database as it was up to the specified date and time, as described at https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL#date. This *must* be in ISO8601 format ("YYYY-MM-DDThh:mm:ssZ"), where both the "T" and "Z" characters must be present.
datetime2	If specified, return the *difference* in the OSM database between datetime and datetime2, where datetime2 > datetime. See https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL#Difference_between_two_dates_(diff).
adiff	If TRUE, query for augmented difference. The result indicates what happened to the modified and deleted OSM objects. Requires datetime(2)*.

`timeout`	It may be necessary to increase this value for large queries, because the server may time out before all data are delivered.
`memsize`	The default memory size for the 'overpass' server in *bytes*; may need to be increased in order to handle large queries.

Value

opq object

Note

Extracting elements by ID requires explicitly specifying the type of element.

References

https://wiki.openstreetmap.org/wiki/Overpass_API/Overpass_QL#By_element_id

See Also

Other queries: `add_osm_feature()`, `add_osm_features()`, `bbox_to_string()`, `getbb()`, `opq()`, `opq_around()`, `opq_csv()`, `opq_enclosing()`, `opq_string()`, `overpass_status()`

Examples

```
## Not run:
id <- c (1489221200, 1489221321, 1489221491)
dat1 <- opq_osm_id (type = "node", id = id) |>
    opq_string () |>
    osmdata_sf ()
dat1$osm_points # the desired nodes
id <- c (136190595, 136190596)
dat2 <- opq_osm_id (type = "way", id = id) |>
    opq_string () |>
    osmdata_sf ()
dat2$osm_lines # the desired ways
dat <- c (dat1, dat2) # The node and way data combined
# All in one (same result as dat)
id <- c (1489221200, 1489221321, 1489221491, 136190595, 136190596)
type <- c ("node", "node", "node", "way", "way")
datAiO <- opq_osm_id (id = id, type = type) |>
    opq_string () |>
    osmdata_sf ()

## End(Not run)
```

opq_string *Convert an overpass query into a text string*

Description

Convert an osmdata query of class opq to a character string query to be submitted to the overpass API.

Usage

```
opq_string(opq)
```

Arguments

opq	An overpass_query object

Value

Character string to be submitted to the overpass API

See Also

Other queries: add_osm_feature(), add_osm_features(), bbox_to_string(), getbb(), opq(), opq_around(), opq_csv(), opq_enclosing(), opq_osm_id(), overpass_status()

Examples

```
## Not run:
q <- opq ("hampi india")
opq_string (q)

## End(Not run)
```

osmdata *osmdata class def*

Description

osmdata class def

Usage

```
osmdata(
  bbox = NULL,
  overpass_call = NULL,
  meta = NULL,
  osm_points = NULL,
  osm_lines = NULL,
  osm_polygons = NULL,
  osm_multilines = NULL,
  osm_multipolygons = NULL
)
```

Arguments

bbox	bounding box
overpass_call	overpass_call
meta	metadata of overpass query, including timestamps and version numbers
osm_points	OSM nodes as **sf** Simple Features Collection of points or **sp** SpatialPointsDataFrame
osm_lines	OSM ways **sf** Simple Features Collection of linestrings or **sp** SpatialLinesDataFrame
osm_polygons	OSM ways as **sf** Simple Features Collection of polygons or **sp** SpatialPolygonsDataFrame
osm_multilines	OSM relations as **sf** Simple Features Collection of multilinestrings or **sp** SpatialLinesDataFrame
osm_multipolygons	OSM relations as **sf** Simple Features Collection of multipolygons or **sp** SpatialPolygonsDataFrame

Note

Class constructor should never be used directly, and is only exported to provide access to the print method

Examples

```
# This function should not need to be called directly!
osmdata ()
```

`osmdata_data_frame` *Return an OSM Overpass query as a data.frame object.*

Description

Return an OSM Overpass query as a data.frame object.

Usage

```
osmdata_data_frame(q, doc, quiet = TRUE, stringsAsFactors = FALSE)
```

Arguments

`q`	An object of class `overpass_query` constructed with opq and add_osm_feature or a string with a valid query, such as `"(node(39.4712701,-0.3841326,39.4713799,-0.3839475);); out;"`. May be be omitted, in which case the attributes of the data.frame will not include the query. See examples below.
`doc`	If missing, `doc` is obtained by issuing the overpass query, `q`, otherwise either the name of a file from which to read data, or an object of class **xml2** returned from osmdata_xml.
`quiet`	suppress status messages.
`stringsAsFactors`	Should character strings in the 'data.frame' be coerced to factors?

Details

If you are not interested in the geometries of the results, it's a good option to query for objects that match the features only and forget about members of the ways and relations. You can achieve this by passing the parameter `body = "tags"` to opq.

Value

A `data.frame` inheriting from `osmdata_data.frame` class with id, type and tags of the the objects from the query.

See Also

Other extract: `osmdata_sc()`, `osmdata_sf()`, `osmdata_sp()`, `osmdata_xml()`

Examples

```
## Not run:
query <- opq ("hampi india") |>
    add_osm_feature (key = "historic", value = "ruins")
# Then extract data from 'Overpass' API
hampi_df <- osmdata_data_frame (query)
attr (hampi_df, "bbox")
```

```
attr (hampi_df, "overpass_call")
attr (hampi_df, "meta")

## End(Not run)

# Complex query as a string (not possible with regular osmdata functions)
q <- '[out:csv(::type, ::id, "name:ca", "wikidata")][timeout:50];
    area[name="Països Catalans"][boundary=political]->.boundaryarea;

    rel(area.boundaryarea)[admin_level=8][boundary=administrative];
    map_to_area -> .all_level_8_areas;

    ( nwr(area.boundaryarea)[amenity=townhall]; >; );
    is_in;
    area._[admin_level=8][boundary=administrative] -> .level_8_areas_with_townhall;

    (.all_level_8_areas; - .level_8_areas_with_townhall;);
    rel(pivot);
    out tags;'

## Not run:
no_townhall <- osmdata_data_frame (q)
no_townhall

## End(Not run)
```

`osmdata_sc` *Return an OSM Overpass query as an* `osmdata_sc` *object in* `silicate` *(*`SC`*) format.*

Description

Return an OSM Overpass query as an `osmdata_sc` object in `silicate` (`SC`) format.

Usage

```
osmdata_sc(q, doc, quiet = TRUE)
```

Arguments

`q`	An object of class `overpass_query` constructed with opq and add_osm_feature or a string with a valid query, such as `"(node(39.4712701,-0.3841326,39.4713799,-0.3839475);); out;"`. 39.4712701,-0.3841326,39.4713799,-0.3839475 May be be omitted, in which case the osmdata object will not include the query. See examples below.
`doc`	If missing, `doc` is obtained by issuing the overpass query, `q`, otherwise either the name of a file from which to read data, or an object of class **xml2** returned from osmdata_xml.
`quiet`	suppress status messages.

Value

An object of class osmdata_sc representing the original OSM hierarchy of nodes, ways, and relations.

Note

The silicate format is currently highly experimental, and recommended for use only if you really know what you're doing.

See Also

Other extract: osmdata_data_frame(), osmdata_sf(), osmdata_sp(), osmdata_xml()

Examples

```
## Not run:
query <- opq ("hampi india") |>
    add_osm_feature (key = "historic", value = "ruins")
# Then extract data from 'Overpass' API
hampi_sc <- osmdata_sc (query)

## End(Not run)

# Complex query as a string (not possible with regular osmdata functions)
q <- '[out:xml][timeout:50];
    area[name="Països Catalans"][boundary=political]->.boundaryarea;

    rel(area.boundaryarea)[admin_level=8][boundary=administrative];
    map_to_area -> .all_level_8_areas;

    ( nwr(area.boundaryarea)[amenity=townhall]; >; );
    is_in;
    area._[admin_level=8][boundary=administrative] -> .level_8_areas_with_townhall;

    (.all_level_8_areas; - .level_8_areas_with_townhall;);
    rel(pivot);
    (._; >;);
    out;'

## Not run:
no_townhall <- osmdata_sc (q)
no_townhall

## End(Not run)
```

---

osmdata_sf *Return an OSM Overpass query as an osmdata object in* **sf** *format.*

Description

Return an OSM Overpass query as an osmdata object in **sf** format.

Usage

```
osmdata_sf(q, doc, quiet = TRUE, stringsAsFactors = FALSE)
```

Arguments

q	An object of class `overpass_query` constructed with opq and add_osm_feature or a string with a valid query, such as `"(node(39.4712701,-0.3841326,39.4713799,-0.3839475);); out;"`. 39.4712701,-0.3841326,39.4713799,-0.3839475 May be be omitted, in which case the osmdata object will not include the query. See examples below.
doc	If missing, doc is obtained by issuing the overpass query, q, otherwise either the name of a file from which to read data, or an object of class **xml2** returned from osmdata_xml.
quiet	suppress status messages.
stringsAsFactors	Should character strings in 'sf' 'data.frame' be coerced to factors?

Value

An object of class `osmdata_sf` with the OSM components (points, lines, and polygons) represented in **sf** format.

Note

In 'dplyr'-type workflows in which the output of this function is piped to other functions, it will generally be necessary to explicitly load the **sf** package into the current workspace with 'library(sf)'.

See Also

Other extract: `osmdata_data_frame()`, `osmdata_sc()`, `osmdata_sp()`, `osmdata_xml()`

Examples

```
## Not run:
query <- opq ("hampi india") |>
    add_osm_feature (key = "historic", value = "ruins")
# Then extract data from 'Overpass' API
hampi_sf <- osmdata_sf (query)

## End(Not run)

# Complex query as a string (not possible with regular osmdata functions)
q <- '[out:xml][timeout:50];
    area[name="Països Catalans"][boundary=political]->.boundaryarea;

    rel(area.boundaryarea)[admin_level=8][boundary=administrative];
    map_to_area -> .all_level_8_areas;
```

```
    ( nwr(area.boundaryarea)[amenity=townhall]; >; );
    is_in;
    area._[admin_level=8][boundary=administrative] -> .level_8_areas_with_townhall;

    (.all_level_8_areas; - .level_8_areas_with_townhall;);
    rel(pivot);
    (._; >;);
    out;'

## Not run:
no_townhall <- osmdata_sf (q)
no_townhall

## End(Not run)
```

`osmdata_sp` *DEPRECATED: Return an OSM Overpass query as an osmdata object in* **sp** *format.*

Description

DEPRECATED: Return an OSM Overpass query as an osmdata object in **sp** format.

Usage

```
osmdata_sp(q, doc, quiet = TRUE)
```

Arguments

`q`	An object of class `overpass_query` constructed with opq and add_osm_feature or a string with a valid query, such as `"(node(39.4712701,-0.3841326,39.4713799,-0.3839475);); out;"`. 39.4712701,-0.3841326,39.4713799,-0.3839475 May be be omitted, in which case the osmdata object will not include the query. See examples below.
`doc`	If missing, `doc` is obtained by issuing the overpass query, `q`, otherwise either the name of a file from which to read data, or an object of class **xml2** returned from osmdata_xml.
`quiet`	suppress status messages.

Value

An object of class `osmdata` with the OSM components (points, lines, and polygons) represented in **sp** format.

See Also

Other extract: `osmdata_data_frame()`, `osmdata_sc()`, `osmdata_sf()`, `osmdata_xml()`

Examples

```
## Not run:
query <- opq ("hampi india") |>
    add_osm_feature (key = "historic", value = "ruins")
# Then extract data from 'Overpass' API
hampi_sp <- osmdata_sp (query)

## End(Not run)

# Complex query as a string (not possible with regular osmdata functions)
q <- '[out:xml][timeout:50];
    area[name="Països Catalans"][boundary=political]->.boundaryarea;

    rel(area.boundaryarea)[admin_level=8][boundary=administrative];
    map_to_area -> .all_level_8_areas;

    ( nwr(area.boundaryarea)[amenity=townhall]; >; );
    is_in;
    area._[admin_level=8][boundary=administrative] -> .level_8_areas_with_townhall;

    (.all_level_8_areas; - .level_8_areas_with_townhall;);
    rel(pivot);
    (._; >;);
    out;'

## Not run:
no_townhall <- osmdata_sp (q)
no_townhall

## End(Not run)
```

`osmdata_xml` *Return an OSM Overpass query in XML format Read an (XML format) OSM Overpass response from a string, a connection, or a raw vector.*

Description

Return an OSM Overpass query in XML format Read an (XML format) OSM Overpass response from a string, a connection, or a raw vector.

Usage

```
osmdata_xml(q, filename, quiet = TRUE, encoding)
```

Arguments

`q`	An object of class `overpass_query` constructed with opq and add_osm_feature or a string with a valid query, such as `"(node(39.4712701,-0.3841326,39.4713799,-0.3839475);); out;"`. See examples below.

filename	If given, OSM data are saved to the named file
quiet	suppress status messages.
encoding	Unless otherwise specified XML documents are assumed to be encoded as UTF-8 or UTF-16. If the document is not UTF-8/16, and lacks an explicit encoding directive, this allows you to supply a default.

Value

An object of class `xml2::xml_document` containing the result of the overpass API query.

Note

Objects of class `xml_document` can be saved as `.xml` or `.osm` files with `xml2::write_xml`.

See Also

Other extract: `osmdata_data_frame()`, `osmdata_sc()`, `osmdata_sf()`, `osmdata_sp()`

Examples

```
## Not run:
query <- opq ("hampi india") |>
    add_osm_feature (key = "historic", value = "ruins")
# Then extract data from 'Overpass' API and save to local file:
osmdata_xml (query, filename = "hampi.osm")

## End(Not run)

# Complex query as a string (not possible with regular osmdata functions)
q <- '[out:xml][timeout:50];
    area[name="Països Catalans"][boundary=political]->.boundaryarea;

    rel(area.boundaryarea)[admin_level=8][boundary=administrative];
    map_to_area -> .all_level_8_areas;

    ( nwr(area.boundaryarea)[amenity=townhall]; >; );
    is_in;
    area._[admin_level=8][boundary=administrative] -> .level_8_areas_with_townhall;

    (.all_level_8_areas; - .level_8_areas_with_townhall;);
    rel(pivot);
    out tags;'

## Not run:
no_townhall <- osmdata_xml (q)
no_townhall

## End(Not run)
```

osm_elevation *osm_elevation*

Description

Add elevation data to a previously-extracted OSM data set, using a pre-downloaded global elevation file from https://srtm.csi.cgiar.org/srtmdata/. Currently only works for SC-class objects returned from osmdata_sc.

Usage

```
osm_elevation(dat, elev_file)
```

Arguments

dat	An SC object produced by osmdata_sc.
elev_file	A vector of one or more character strings specifying paths to .tif files (or anything that **terra** can read) containing global elevation data. .zip files will be uncompressed.

Value

A modified version of the input dat with an additional z_ column appended to the vertices.

See Also

Other transform: osm_poly2line(), trim_osmdata(), unique_osmdata(), unname_osmdata_sf()

Examples

```
## Not run:
query <- opq ("omaha nebraska") |>
    add_osm_feature (key = "highway")
# Elevation can only be applied to \pkg{silicate} 'SC'-class data:
dat <- osmdata_sc (query)
dat$vertex
# The vertex table will have columns ("x_", "y_", "vertex_"). Then
# download elevation data from \url{https://srtm.csi.cgiar.org/srtmdata/}
# (or elsewhere), and add elevation column, "z_" with:
dat <- osm_elevation (dat, elev_file = "/path/to/elevation/data.tiff")

## End(Not run)
```

`osm_lines` *Extract all* `osm_lines` *from an* `osmdata_sf` *object*

Description

If `id` is of a point object, `osm_lines` will return all lines containing that point. If `id` is of a line or polygon object, `osm_lines` will return all lines which intersect the given line or polygon.

Usage

```
osm_lines(dat, id)
```

Arguments

`dat`	An object of class `osmdata_sf`
`id`	OSM identification of one or more objects for which lines are to be extracted

Value

An **sf** Simple Features Collection of linestrings

See Also

Other search: `osm_multilines()`, `osm_multipolygons()`, `osm_points()`, `osm_polygons()`

Examples

```
## Not run:
dat <- opq ("hengelo nl") |>
    add_osm_feature (key = "highway") |>
    osmdata_sf ()
bus <- dat$osm_points [which (dat$osm_points$highway == "bus_stop"), ] |>
    rownames () # all OSM IDs of bus stops
osm_lines (dat, bus) # all highways containing bus stops

# All lines which intersect with Piccadilly Circus in London, UK
dat <- opq ("Fitzrovia London") |>
    add_osm_feature (key = "highway") |>
    osmdata_sf ()
i <- which (dat$osm_polygons$name == "Piccadilly Circus")
id <- rownames (dat$osm_polygons [i, ])
osm_lines (dat, id)

## End(Not run)
```

osm_multilines *Extract all* osm_multilines *from an* osmdata_sf *object*

Description

id must be of an osm_points or osm_lines object (and can not be the id of an osm_polygons object because multilines by definition contain no polygons. osm_multilines returns any multiline object(s) which contain the object specified by id.

Usage

```
osm_multilines(dat, id)
```

Arguments

dat	An object of class osmdata_sf
id	OSM identification of one of more objects for which multilines are to be extracted

Value

An **sf** Simple Features Collection of multilines

See Also

Other search: osm_lines(), osm_multipolygons(), osm_points(), osm_polygons()

Examples

```
## Not run:
dat <- opq ("London UK") |>
    add_osm_feature (key = "name", value = "Thames", exact = FALSE) |>
    osmdata_sf ()
# Get ids of lines called "The Thames":
id <- rownames (dat$osm_lines [which (dat$osm_lines$name == "The Thames"), ])
# and find all multilinestring objects which include those lines:
osm_multilines (dat, id)
# Now note that
nrow (dat$osm_multilines) # = 24 multiline objects
nrow (osm_multilines (dat, id)) # = 1 - the recursive search selects the
# single multiline containing "The Thames"

## End(Not run)
```

`osm_multipolygons` *Extract all* `osm_multipolygons` *from an* `osmdata_sf` *object*

Description

`id` must be of an `osm_points`, `osm_lines`, or `osm_polygons` object. `osm_multipolygons` returns any multipolygon object(s) which contain the object specified by `id`.

Usage

```
osm_multipolygons(dat, id)
```

Arguments

`dat`	An object of class `osmdata_sf`
`id`	OSM identification of one or more objects for which multipolygons are to be extracted

Value

An **sf** Simple Features Collection of multipolygons

See Also

Other search: `osm_lines()`, `osm_multilines()`, `osm_points()`, `osm_polygons()`

Examples

```
# find all multipolygons which contain the single polygon called
# "Chiswick Eyot" (which is an island).
## Not run:
dat <- opq ("London UK") |>
    add_osm_feature (key = "name", value = "Thames", exact = FALSE) |>
    osmdata_sf ()
index <- which (dat$osm_multipolygons$name == "Chiswick Eyot")
id <- rownames (dat$osm_polygons [id, ])
osm_multipolygons (dat, id)
# That multipolygon is the Thames itself, but note that
nrow (dat$osm_multipolygons) # = 14 multipolygon objects
nrow (osm_multipolygons (dat, id)) # = 1 - the main Thames multipolygon

## End(Not run)
```

osm_points *Extract all* osm_points *from an* osmdata_sf *object*

Description

Extract all osm_points from an osmdata_sf object

Usage

```
osm_points(dat, id)
```

Arguments

dat	An object of class osmdata_sf
id	OSM identification of one or more objects for which points are to be extracted

Value

An **sf** Simple Features Collection of points

See Also

Other search: osm_lines(), osm_multilines(), osm_multipolygons(), osm_polygons()

Examples

```
## Not run:
tr <- opq ("trentham australia") |> osmdata_sf ()
coliban <- tr$osm_lines [which (tr$osm_lines$name == "Coliban River"), ]
pts <- osm_points (tr, rownames (coliban)) # all points of river
# the waterfall point:
waterfall <- pts [which (pts$waterway == "waterfall"), ]

## End(Not run)
```

osm_poly2line *Convert osmdata polygons into lines*

Description

Street networks downloaded with add_osm_object(key = "highway") will store any circular highways in osm_polygons. this function combines those with the osm_lines component to yield a single **sf** data.frame of all highways, whether polygonal or not.

Usage

```
osm_poly2line(osmdat)
```

Arguments

osmdat	An osmdata_sf object.

Value

Modified version of same object with all osm_polygons objects merged into osm_lines.

Note

The osm_polygons field is retained, with those features also repeated as LINESTRING objects in osm_lines.

See Also

Other transform: osm_elevation(), trim_osmdata(), unique_osmdata(), unname_osmdata_sf()

Examples

```
## Not run:
query <- opq ("colchester uk") |>
    add_osm_feature (key = "highway")
# Then extract data from 'Overpass' API
dat <- osmdata_sf (query)
# colchester has lots of roundabouts, and these are stored in 'osm_polygons'
# rather than 'osm_lines'. The former can be merged with the latter by:
dat2 <- osm_poly2line (dat)

## End(Not run)
# 'dat2' will have more lines than 'dat', but the same number of polygons
# (they are left unchanged.)
```

---

osm_polygons *Extract all* osm_polygons *from an* osmdata_sf *object*

Description

If id is of a point object, osm_polygons will return all polygons containing that point. If id is of a line or polygon object, osm_polygons will return all polygons which intersect the given line or polygon.

Usage

```
osm_polygons(dat, id)
```

Arguments

dat	An object of class osmdata_sf
id	OSM identification of one or more objects for which polygons are to be extracted

Value

An **sf** Simple Features Collection of polygons

See Also

Other search: osm_lines(), osm_multilines(), osm_multipolygons(), osm_points()

Examples

```
# Extract polygons which intersect Conway Street in London
## Not run:
dat <- opq ("Marylebone London") |>
    add_osm_feature (key = "highway") |>
    osmdata_sf ()
conway <- which (dat$osm_lines$name == "Conway Street")
id <- rownames (dat$osm_lines [conway, ])
osm_polygons (dat, id)

## End(Not run)
```

---

overpass_status *Retrieve status of the Overpass API*

Description

Retrieve status of the Overpass API

Usage

```
overpass_status(quiet = FALSE)
```

Arguments

quiet	if FALSE display a status message

Value

an invisible list of whether the API is available along with the text of the message from Overpass and the timestamp of the next available slot

See Also

Other queries: add_osm_feature(), add_osm_features(), bbox_to_string(), getbb(), opq(), opq_around(), opq_csv(), opq_enclosing(), opq_osm_id(), opq_string()

Examples

```
## Not run:
overpass_status ()

## End(Not run)
```

`set_overpass_url` *set_overpass_url*

Description

Set the URL of the specified overpass API. Possible APIs with global coverage are:

- "https://overpass-api.de/api/interpreter" (default)
- "https://overpass.kumi.systems/api/interpreter"
- "https://overpass.osm.rambler.ru/cgi/interpreter"
- "https://api.openstreetmap.fr/oapi/interpreter"
- "https://overpass.osm.vi-di.fr/api/interpreter"

Additional APIs with limited local coverage include:

- "https://overpass.osm.ch/api/interpreter" (Switzerland)
- "https://overpass.openstreetmap.ie/api/interpreter" (Ireland)

Usage

```
set_overpass_url(overpass_url)
```

Arguments

`overpass_url`	The desired overpass API URL

Details

For further details, see https://wiki.openstreetmap.org/wiki/Overpass_API

Value

The overpass API URL

See Also

`get_overpass_url()`

Other overpass: `get_overpass_url()`

trim_osmdata *trim_osmdata*

Description

Trim an osmdata object to within a bounding polygon

Usage

```
trim_osmdata(dat, bb_poly, exclude = TRUE)
```

Arguments

dat	An osmdata object returned from osmdata_sf() or osmdata_sc().
bb_poly	An sf or sfc object, or matrix representing a bounding polygon. Can be obtained with getbb(..., format_out = "polygon") or getbb(..., format_out = "sf_polygon") (and possibly selected from resultant list where multiple polygons are returned).
exclude	If TRUE, objects are trimmed exclusively, only retaining those strictly within the bounding polygon; otherwise all objects which partly extend within the bounding polygon are retained.

Value

A trimmed version of dat, reduced only to those components lying within the bounding polygon.

Note

It will generally be necessary to pre-load the **sf** package for this function to work correctly.

Caution is advised when using polygons obtained from Nominatim via getbb(..., format_out = "polygon"|"sf_polygon"). These shapes can be outdated and thus could cause the trimming operation to not give results expected based on the current state of the OSM data.

To reduce the downloaded data from Overpass, you can do the trimming in the server-side using getbb(..., format_out = "osm_type_id") (see examples).

See Also

Other transform: osm_elevation(), osm_poly2line(), unique_osmdata(), unname_osmdata_sf()

Examples

```
## Not run:
bb <- getbb ("colchester uk")
query <- opq (bb) |>
    add_osm_feature (key = "highway")
# Then extract data from 'Overpass' API
dat <- osmdata_sf (query, quiet = FALSE)
```

```
# Then get bounding *polygon* for Colchester, as opposed to rectangular
# bounding box, and use that to trim data within that polygon:
bb_pol <- getbb ("colchester uk", format_out = "polygon")
library (sf) # required for this function to work
dat_tr <- trim_osmdata (dat, bb_pol)
bb_sf <- getbb ("colchester uk", format_out = "sf_polygon")
class (bb_sf) # sf data.frame
dat_tr <- trim_osmdata (dat, bb_sf)
bb_sp <- as (bb_sf, "Spatial")
class (bb_sp) # SpatialPolygonsDataFrame
dat_tr <- trim_osmdata (dat, bb_sp)

# Server-side trimming equivalent
bb <- getbb ("colchester uk", format_out = "osm_type_id")
query <- opq (bb) |>
    add_osm_feature (key = "highway")
dat <- osmdata_sf (query, quiet = FALSE)

## End(Not run)
```

`unique_osmdata` *unique_osmdata*

Description

Reduce the components of an osmdata object in 'sf' or 'sp' form (that is, obtained from `osmdata_sf()` or `osmdata_sp()` to only unique items of each type. That is, reduce `$osm_points` to only those points not present in other objects (lines, polygons, etc.); reduce `$osm_lines` to only those lines not present in multiline objects; and reduce `$osm_polygons` to only those polygons not present in multipolygon objects. This renders an osmdata object more directly compatible with typical output of **sf**.

Usage

```
unique_osmdata(dat)
```

Arguments

`dat`	An object of class `osmdata_sf` or `osmdata_sp`

Value

Equivalent object reduced to only unique objects of each type

See Also

Other transform: `osm_elevation()`, `osm_poly2line()`, `trim_osmdata()`, `unname_osmdata_sf()`

Examples

```
## Not run:
query <- opq ("colchester uk") |>
    add_osm_feature (key = "highway")
# Then extract data from 'Overpass' API
dat <- osmdata_sf (query)
dat
# Then reduce to unique items of each type only:
dat <- unique_osmdata (dat)
dat

## End(Not run)
# And objects of each type (points, line, polygons, and so on) now have
# fewer members.
```

`unname_osmdata_sf` *unname_osmdata_sf*

Description

Remove names from `osmdata` geometry objects, for cases in which these cause issues, particularly with plotting, such as https://github.com/rstudio/leaflet/issues/631, or https://github.com/r-spatial/sf/issues/1177. Note that removing these names also removes any ability to inter-relate the different components of an `osmdata` object, so use of this function is only recommended to resolve issues such as those linked to above.

Usage

```
unname_osmdata_sf(x)
```

Arguments

`x`	An 'osmdata_sf' object returned from function of same name

Value

Same object, yet with no row names on geometry objects.

See Also

Other transform: `osm_elevation()`, `osm_poly2line()`, `trim_osmdata()`, `unique_osmdata()`

Examples

```
## Not run:
query <- opq ("hampi india") |>
    add_osm_feature (key = "historic", value = "ruins")
# Then extract data from 'Overpass' API
hampi_sf <- osmdata_sf (query)
```

```
# Remove rownames from all geometry metrices:
hampi_clean <- unname_osmdata_sf (hampi_sf)

# All coordinate matrices include rownames with OSM ID values:
head (as.matrix (hampi_sf$osm_lines$geometry [[1]]))
# But 'unname_osmdata_sf' removes both row and column names:
head (as.matrix (hampi_clean$osm_lines$geometry [[1]]))

## End(Not run)
```

Index