

# Package: osmextract (via r-universe)

November 22, 2024

**Type** Package

**Title** Download and Import Open Street Map Data Extracts

**Version** 0.5.2.9000

**Description** Match, download, convert and import Open Street Map data extracts obtained from several providers.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.2

**Roxygen** list(markdown = TRUE)

**URL** <https://docs.ropensci.org/osmextract/>,  
<https://github.com/ropensci/osmextract>

**BugReports** <https://github.com/ropensci/osmextract/issues>

**Depends** R (>= 3.6.0)

**Imports** sf (>= 0.8.1), utils, tools, httr, jsonlite

**Suggests** testthat (>= 3.0.2), knitr, rmarkdown, covr, withr

**VignetteBuilder** knitr

**Language** en-GB

**Config/testthat/edition** 3

**Config/build/copy-method** link

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

**Repository** <https://ropensci.r-universe.dev>

**RemoteUrl** <https://github.com/ropensci/osmextract>

**RemoteRef** master

**RemoteSha** 35f1434c0091ba1534e8801e760b559cc940bb85

## Contents

bbbike_zones	2
geofabrik_zones	3
oe_clean	4
oe_download	5
oe_download_directory	6
oe_find	7
oe_get	9
oe_get_boundary	14
oe_get_keys	15
oe_get_network	19
oe_match	21
oe_match_pattern	24
oe_providers	26
oe_read	26
oe_search	29
oe_update	30
oe_vectortranslate	32
openstreetmap_fr_zones	36
read_poly	37
test_zones	38

<b>Index</b>	<b>39</b>
--------------	-----------

---

bbbike_zones	<i>An sf object of geographical zones taken from bbbike.org</i>
--------------	---

---

### Description

Start bicycle routing for... everywhere!

### Usage

bbbike\_zones

### Format

An sf object with 237 rows and 6 columns:

**name** The, usually English, long-form name of the city.

**pbf** Link to the latest .osm.pbf file for this region.

**pbf\_file\_size** Size of the pbf file in bytes.

**id** A unique identifier. It contains letters, numbers and potentially the characters "-" and "/".

**level** An integer code always equal to 3 (since the bbbike data represent non-hierarchical geographical zones). This is used only for matching operations in case of spatial input. The oe\_\* functions will select the geographical area closest to the input place with the highest "level". See [geofabrik\\_zones](#) for an example of a (proper) hierarchical structure.

**geometry** The sfg for that geographical region, rectangular. See also `oe_get_boundary()` to extract the proper geographical boundaries.

### Details

An sf object containing the URLs, names, and file\_size of the OSM extracts stored at <https://download.bbbike.org/osm/bbbike/>.

### Source

<https://download.bbbike.org/osm/>

### See Also

Other provider's-database: [geofabrik\\_zones](#), [openstreetmap\\_fr\\_zones](#)

---

geofabrik\_zones

*An sf object of geographical zones taken from geofabrik.de*

---

### Description

An sf object containing the URLs, names and file-sizes of the OSM extracts stored at <https://download.geofabrik.de/>. You can read more details about these data at the following link: <https://download.geofabrik.de/technical.html>.

### Usage

```
geofabrik_zones
```

### Format

An sf object with 476 rows and 9 columns:

**id** A unique identifier. It contains letters, numbers and potentially the characters "-" and "/".

**name** The, usually English, long-form name of the area.

**parent** The identifier of the next larger excerpts that contains this one, if present.

**level** An integer code between 1 and 4. If level = 1, then the zone corresponds to one of the continents (Africa, Antarctica, Asia, Australia and Oceania, Central America, Europe, North America, and South America) or the Russian Federation. If level = 2, then the zone corresponds to the continent's subregions (i.e. the countries such as Italy, Great Britain, Spain, USA, Mexico, Belize, Morocco, Peru and so on). There are also some exceptions that correspond to the Special Sub Regions (according to the Geofabrik definition), which are: South Africa (includes Lesotho), Alps, Britain and Ireland, Germany + Austria + Switzerland, US Midwest, US Northeast, US Pacific, US South, US West, and all US states. Level = 3L corresponds to the subregions of each state (or each level 2 zone). For example, the West Yorkshire, which is a subregion of England, is a level 3 zone. Finally, level = 4 correspond to the subregions of the third level and it is mainly related to some small areas in Germany. This field is used only for matching operations in case of spatial input.

**iso3166-1\_alpha2** A character vector of two-letter **ISO3166-1 codes**. This will be set on the smallest extract that still fully (or mostly) contains the entity with that code; e.g. the code "DE" will be given for the Germany extract and not for Europe even though Europe contains Germany. If an extract covers several countries and no per-country extracts are available (e.g. Israel and Palestine), then several ISO codes will be given (such as "PS IL" for "Palestine and Israel").

**iso3166\_2** A character vector of usually five-character **ISO3166-2 codes**. The same rules as above apply. Some entities have both an *iso3166-1* and *iso3166-2* code. For example, the *iso3166\_2* code of each US State is "US - " plus the code of the state.

**pbf** Link to the latest .osm.pbf file for this region.

**pbf\_file\_size** Size of the .pbf file in bytes.

**geometry** The sfg for that geographical region. These are not the country boundaries, but a buffer around countries. Check `oe_get_boundary()` to extract the geographical boundaries.

### Source

<https://download.geofabrik.de/>

### See Also

Other provider's-database: [bbbike\\_zones](#), [openstreetmap\\_fr\\_zones](#)

---

oe\_clean

*Clean download directory*

---

### Description

This functions is a wrapper around `unlink()` that can be used to delete all .osm.pbf and .gpkg files in a given directory.

### Usage

```
oe_clean(download_directory = oe_download_directory(), force = FALSE)
```

### Arguments

`download_directory`

The directory where the .osm.pbf and .gpkg files are saved. Default value is `oe_download_directory()`.

`force`

Internal option. It can be used to skip the checks run at the beginning of the function and force the removal of all pbf/gpkg files.

### Value

The same as `unlink()`.

**Examples**

```
# Warning: the following removes all files in oe_download_directory()
## Not run:
oe_clean()
## End(Not run)
```

---

oe_download	<i>Download a file given a url</i>
-------------	------------------------------------

---

**Description**

This function is used to download a file given a URL. It focuses on OSM extracts with .osm.pbf format stored by one of the providers implemented in the package. The URL is specified through the parameter `file_url`.

**Usage**

```
oe_download(
  file_url,
  provider = NULL,
  file_basename = basename(file_url),
  download_directory = oe_download_directory(),
  file_size = NA,
  force_download = FALSE,
  max_file_size = 5e+08,
  quiet = FALSE
)
```

**Arguments**

<code>file_url</code>	A URL pointing to a .osm.pbf file that should be downloaded.
<code>provider</code>	Which provider stores the file? If NULL (the default), it may be inferred from the URL, but it must be specified for non-standard cases. See details and examples.
<code>file_basename</code>	The basename of the file. The default behaviour is to auto-generate it from the URL using <code>basename()</code> .
<code>download_directory</code>	Directory to store the file containing OSM data?.
<code>file_size</code>	How big is the file? Optional. NA by default. If it's bigger than <code>max_file_size</code> and the function is run in interactive mode, then an interactive menu is displayed, asking for permission for downloading the file.
<code>force_download</code>	Should the .osm.pbf file be updated even if it has already been downloaded? FALSE by default. This parameter is used to update old .osm.pbf files.
<code>max_file_size</code>	The maximum file size to download without asking in interactive mode. Default: 5e+8, half a gigabyte.
<code>quiet</code>	Boolean. If FALSE, the function prints informative messages. Starting from sf version 0.9.6, if quiet is equal to FALSE, then vectortranslate operations will display a progress bar.

## Details

This function runs several checks before actually downloading a new file to avoid overloading the OSM providers. The first step is the definition of the file's path associated to the input `file_url`. The path is created by pasting together the `download_directory`, the name of chosen provider (which may be inferred from the URL) and the `basename()` of the URL. For example, if `file_url` is equal to `"https://download.geofabrik.de/europe/italy-latest.osm.pbf"`, and `download_directory` = `"/tmp"`, then the path is built as `"/tmp/geofabrik_italy-latest.osm.pbf"`. Thereafter, the function checks the existence of that file and, if it finds it, then it returns the path. The parameter `force_download` is used to modify this behaviour. If there is no file associated with the new path, then the function downloads a new file using `download.file()` with mode = `"wb"`, and, again, it returns the path.

## Value

A character string representing the file's path.

## Examples

```
(its_match = oe_match("ITS Leeds", quiet = TRUE))

## Not run:
oe_download(
  file_url = its_match$url,
  file_size = its_match$file_size,
  provider = "test",
  download_directory = tempdir()
)
iow_url = oe_match("Isle of Wight")
oe_download(
  file_url = iow_url$url,
  file_size = iow_url$file_size,
  download_directory = tempdir()
)
Sucre_url = oe_match("Sucre", provider = "bbbike")
oe_download(
  file_url = Sucre_url$url,
  file_size = Sucre_url$file_size,
  download_directory = tempdir()
)
## End(Not run)
```

---

`oe_download_directory` *Return the download directory used by the package*

---

## Description

By default, the download directory is equal to `tempdir()`. You can set a persistent download directory by adding the following command to your `.Renv` file (e.g. with `edit_r_env` function in `usethis` package): `OSMEXT_DOWNLOAD_DIRECTORY=/path/to/osm/data`.

**Usage**

```
oe_download_directory()
```

**Value**

A character vector representing the path for the download directory used by the package.

**Examples**

```
oe_download_directory()
```

---

oe_find	<i>Get the path of .pbf and .gpkg files associated with an input OSM extract</i>
---------	--

---

**Description**

This function takes a place name and returns the path of .pbf/.gpkg files associated with it.

**Usage**

```
oe_find(
  place,
  provider = "geofabrik",
  download_directory = oe_download_directory(),
  download_if_missing = FALSE,
  return_pbf = TRUE,
  return_gpkg = TRUE,
  quiet = FALSE,
  ...
)
```

**Arguments**

place	Description of the geographical area that should be matched with a .osm.pbf file. Can be either a length-1 character vector, an sf/sfc/bbox object with any CRS, or a numeric vector of coordinates with length 2. In the last case, it is assumed that the EPSG code is 4326 specified as c(LON, LAT), while you can use any CRS with sf/sfc/bbox objects. See Details and Examples in <a href="#">oe_match()</a> .
provider	Which provider should be used to download the data? Available providers can be browsed with <a href="#">oe_providers()</a> . For <a href="#">oe_get()</a> and <a href="#">oe_match()</a> , if place is equal to ITS Leeds, then provider is internally set equal to "test". This is just for simple examples and internal tests.
download_directory	Directory where the files downloaded by osmextract are stored. By default it is equal to <a href="#">oe_download_directory()</a> .

download_if_missing	Attempt to download the file if it cannot be found? FALSE by default.
return_pbf	Logical of length 1. If TRUE, the function returns the path of the pbf file that matches the input place.
return_gpkg	Logical of length 1. If TRUE, the function returns the path of the gpkg file that matches the input place.
quiet	Boolean. If FALSE, the function prints informative messages. Starting from sf version 0.9.6, if quiet is equal to FALSE, then vectortranslate operations will display a progress bar.
...	Extra arguments that are passed to <code>oe_match()</code> and <code>oe_get()</code> . Please note that you cannot modify the argument <code>download_only</code> .

### Details

The matching between the existing files (saved in the directory specified by `download_directory` parameter) and the input `place` is performed using `list.files()`, setting the `pattern` argument equal to the basename of the URL associated to the input `place`. For example, if you specify `place = "Isle of Wight"`, then the input is matched (via `oe_match()`) with the URL of Isle of Wight's `.osm.pbf` file, and the files are selected using a pattern equal to the basename of that URL.

If there is no file in the `download_directory` that can be matched with the basename of the URL and `download_if_missing` parameter is equal to TRUE, then the function tries to download and read a new file from the chosen provider (geofabrik is the default provider). If `download_if_missing` parameter is equal to FALSE (default value), then the function stops with an error.

By default, this function returns the path of `.pbf` and `.gpkg` files associated with the input `place` (if any). You can exclude one of the two formats setting the arguments `return_pbf` or `return_gpkg` to FALSE.

### Value

A character vector of length one (or two) representing the path(s) of the `.pbf/.gpkg` files associated with the input `place`. The files are sorted in alphabetical order which implies that if both formats are present in the `download_directory`, then the `.gpkg` file is returned first.

### Examples

```
# Copy the ITS file to tempdir() to make sure that the examples do not
# require internet connection. You can skip the next 4 lines (and start
# directly with oe_get_keys) when running the examples locally.

res = file.copy(
  from = system.file("its-example.osm.pbf", package = "osmextract"),
  to = file.path(tempdir(), "test_its-example.osm.pbf"),
  overwrite = TRUE
)
res = oe_get("ITS Leeds", quiet = TRUE, download_directory = tempdir())
oe_find("ITS Leeds", provider = "test", download_directory = tempdir())
oe_find(
  "ITS Leeds", provider = "test",
  download_directory = tempdir(), return_gpkg = FALSE
)
```



```

)

## Not run:
oe_find("Isle of Wight", download_directory = tempdir())
oe_find("Malta", download_if_missing = TRUE, download_directory = tempdir())
oe_find(
  "Leeds",
  provider = "bbbike",
  download_if_missing = TRUE,
  download_directory = tempdir(),
  return_pbf = FALSE
)
## End(Not run)

# Remove .pbf and .gpkg files in tempdir
oe_clean(tempdir())

```

---

oe_get	<i>Find, download, translate and read OSM extracts from several providers</i>
--------	---

---

## Description

This function is used to find, download, translate, and read OSM extracts obtained from several providers. It is a wrapper around `oe_match()` and `oe_read()`. Check the introductory vignette, the examples, and the help pages of the wrapped functions to understand the details behind all parameters.

## Usage

```

oe_get(
  place,
  layer = "lines",
  ...,
  provider = "geofabrik",
  match_by = "name",
  max_string_dist = 1,
  level = NULL,
  download_directory = oe_download_directory(),
  force_download = FALSE,
  max_file_size = 5e+08,
  vectortranslate_options = NULL,
  osmconf_ini = NULL,
  extra_tags = NULL,
  force_vectortranslate = FALSE,
  boundary = NULL,
  boundary_type = c("spat", "clipsrc"),
  download_only = FALSE,

```

```

    skip_vectortranslate = FALSE,
    never_skip_vectortranslate = FALSE,
    quiet = FALSE
  )

```

### Arguments

place	Description of the geographical area that should be matched with a <code>.osm.pbf</code> file. Can be either a length-1 character vector, an <code>sf/sfc/bbox</code> object with any CRS, or a numeric vector of coordinates with length 2. In the last case, it is assumed that the EPSG code is 4326 specified as <code>c(LON, LAT)</code> , while you can use any CRS with <code>sf/sfc/bbox</code> objects. See Details and Examples in <code>oe_match()</code> .
layer	Which layer should be read in? Typically points, lines (the default), <code>multilinestrings</code> , <code>multipolygons</code> or <code>other_relations</code> . If you specify an ad-hoc query using the argument <code>query</code> (see introductory vignette and examples), then <code>oe_get()</code> and <code>oe_read()</code> will read the layer specified in the query and ignore <code>layer</code> argument. See also <a href="#">#122</a> .
...	(Named) arguments that will be passed to <code>sf::st_read()</code> , like <code>query</code> , <code>wkt_filter</code> or <code>stringsAsFactors</code> . Check the introductory vignette to understand how to create your own (SQL-like) queries.
provider	Which provider should be used to download the data? Available providers can be browsed with <code>oe_providers()</code> . For <code>oe_get()</code> and <code>oe_match()</code> , if <code>place</code> is equal to <code>ITS Leeds</code> , then <code>provider</code> is internally set equal to <code>"test"</code> . This is just for simple examples and internal tests.
match_by	Which column of the provider's database should be used for matching the input <code>place</code> with a <code>.osm.pbf</code> file? The default is <code>"name"</code> . Check Details and Examples in <code>oe_match()</code> to understand how this parameter works. Ignored when <code>place</code> is not a character vector since, in that case, the matching is performed through a spatial operation.
max_string_dist	Numerical value greater or equal than 0. What is the maximum distance in fuzzy matching (i.e. Approximate String Distance, see <code>adist()</code> ) between input <code>place</code> and <code>match_by</code> column that can be tolerated before testing alternative providers or looking for geographical matching with Nominatim API? This parameter is set equal to 0 if <code>match_by</code> is equal to <code>iso3166_1_alpha2</code> or <code>iso3166_2</code> . Check Details and Examples in <code>oe_match()</code> to understand why this parameter is important. Ignored when <code>place</code> is not a character vector since, in that case, the matching is performed through a spatial operation.
level	An integer representing the desired hierarchical level in case of spatial matching. For the <code>geofabrik</code> provider, for example, 1 corresponds with continent-level datasets, 2 for countries, 3 corresponds to regions and 4 to subregions. Hence, we could approximately say that smaller administrative units correspond to bigger levels. If <code>NULL</code> , the default, the <code>oe_*</code> functions will select the highest available level. See Details and Examples in <code>oe_match()</code> .
download_directory	Directory to store the file containing OSM data?.

<code>force_download</code>	Should the <code>.osm.pbf</code> file be updated even if it has already been downloaded? FALSE by default. This parameter is used to update old <code>.osm.pbf</code> files.
<code>max_file_size</code>	The maximum file size to download without asking in interactive mode. Default: $5e+8$ , half a gigabyte.
<code>vectortranslate_options</code>	Options passed to the <code>sf::gdal_utils()</code> argument options. Set by default. Check details in the introductory vignette and the help page of <code>oe_vectortranslate()</code> .
<code>osmconf_ini</code>	The configuration file. See documentation at <a href="http://gdal.org">gdal.org</a> . Check details in the introductory vignette and the help page of <code>oe_vectortranslate()</code> . Set by default.
<code>extra_tags</code>	Which additional columns, corresponding to OSM tags, should be in the resulting dataset? NULL by default. Check the introductory vignette and the help pages of <code>oe_vectortranslate()</code> and <code>oe_get_keys()</code> . Ignored when <code>osmconf_ini</code> is not NULL.
<code>force_vectortranslate</code>	Boolean. Force the original <code>.pbf</code> file to be translated into a <code>.gpkg</code> file, even if a <code>.gpkg</code> with the same name already exists? FALSE by default. If tags in <code>extra_tags</code> match data in previously translated <code>.gpkg</code> files no translation occurs (see <a href="#">#173</a> for details). Check the introductory vignette and the help page of <code>oe_vectortranslate()</code> .
<code>boundary</code>	An <code>sf/sfc/bbox</code> object that will be used to create a spatial filter during the <code>vectortranslate</code> operations. The type of filter can be chosen using the argument <code>boundary_type</code> .
<code>boundary_type</code>	A character vector of length 1 specifying the type of spatial filter. The spat filter selects only those features that intersect a given area, while <code>clipsrc</code> also clips the geometries. Check the examples and also <a href="#">here</a> for more details.
<code>download_only</code>	Boolean. If TRUE, then the function only returns the path where the matched file is stored, instead of reading it. FALSE by default.
<code>skip_vectortranslate</code>	Boolean. If TRUE, then the function skips all <code>vectortranslate</code> operations and it reads (or simply returns the path) of the <code>.osm.pbf</code> file. FALSE by default.
<code>never_skip_vectortranslate</code>	Boolean. This is used in case the user passed its own <code>.ini</code> file or <code>vectortranslate</code> options (since, in those case, it's too difficult to determine if an existing <code>.gpkg</code> file was generated following the same options.)
<code>quiet</code>	Boolean. If FALSE, the function prints informative messages. Starting from <code>sf</code> version <a href="#">0.9.6</a> , if <code>quiet</code> is equal to FALSE, then <code>vectortranslate</code> operations will display a progress bar.

## Details

The algorithm that we use for importing an OSM extract data into R is divided into 4 steps: 1) match the input place with the url of a `.pbf` file; 2) download the `.pbf` file; 3) convert it into `.gpkg` format and 4) read-in the `.gpkg` file. The function `oe_match()` is used to perform the first operation and the function `oe_read()` (which is a wrapper around `oe_download()`, `oe_vectortranslate()` and `sf::st_read()`) performs the other three operations.

**Value**

An sf object.

**See Also**

[oe\\_match\(\)](#), [oe\\_download\(\)](#), [oe\\_vectortranslate\(\)](#), and [oe\\_read\(\)](#).

**Examples**

```
# Copy ITS file to tempdir so that the examples do not require internet
# connection. You can skip the next 4 lines when running the examples
# locally.

its_pbf = file.path(tempdir(), "test_its-example.osm.pbf")
file.copy(
  from = system.file("its-example.osm.pbf", package = "osmextract"),
  to = its_pbf,
  overwrite = TRUE
)

# Match, download (not really) and convert OSM extracts associated to a simple test.
its = oe_get("ITS Leeds", download_directory = tempdir())
class(its)
unique(sf::st_geometry_type(its))

# Get another layer from ITS Leeds extract
its_points = oe_get("ITS Leeds", layer = "points", download_directory = tempdir())
unique(sf::st_geometry_type(its_points))

# Get the .osm.pbf and .gpkg files paths
oe_get(
  "ITS Leeds", download_only = TRUE, quiet = TRUE,
  download_directory = tempdir()
)
oe_get(
  "ITS Leeds", download_only = TRUE, skip_vectortranslate = TRUE,
  quiet = TRUE, download_directory = tempdir()
)
# See also ?oe_find()

# Add additional tags
its_with_oneway = oe_get(
  "ITS Leeds", extra_tags = "oneway",
  download_directory = tempdir()
)
names(its_with_oneway)
table(its_with_oneway$oneway, useNA = "ifany")

# Use the query argument to get only oneway streets:
q = "SELECT * FROM 'lines' WHERE oneway == 'yes'"
its_oneway = oe_get("ITS Leeds", query = q, download_directory = tempdir())
its_oneway[, c(1, 3, 9)]
```

```

# Apply a spatial filter during the vectortranslate operations
its_poly = sf::st_sfc(
  sf::st_polygon(
    list(rbind(
      c(-1.55577, 53.80850),
      c(-1.55787, 53.80926),
      c(-1.56096, 53.80891),
      c(-1.56096, 53.80736),
      c(-1.55675, 53.80658),
      c(-1.55495, 53.80749),
      c(-1.55577, 53.80850)
    ))
  ),
  crs = 4326
)
its_spat = oe_get("ITS Leeds", boundary = its_poly, download_directory = tempdir())
its_clipped = oe_get(
  "ITS Leeds", boundary = its_poly, boundary_type = "clipsrc",
  quiet = TRUE, download_directory = tempdir()
)

plot(sf::st_geometry(its), reset = FALSE, col = "lightgrey")
plot(sf::st_boundary(its_poly), col = "black", add = TRUE)
plot(sf::st_boundary(sf::st_as_sfc(sf::st_bbox(its_poly))), col = "black", add = TRUE)
plot(sf::st_geometry(its_spat), add = TRUE, col = "darkred")
plot(sf::st_geometry(its_clipped), add = TRUE, col = "orange")

# More complex examples
## Not run:
west_yorkshire = oe_get("West Yorkshire")
# If you run it again, the function will not download the file
# or convert it again
west_yorkshire = oe_get("West Yorkshire")
# Match with place name
oe_get("Milan") # Warning: the .pbf file is 400MB
oe_get("Vatican City") # Check all providers
oe_get("Zurich") # Use Nominatim API for geolocating places

# Match with coordinates (any EPSG)
milan_duomo = sf::st_sfc(sf::st_point(c(1514924, 5034552)), crs = 3003)
oe_get(milan_duomo, quiet = FALSE) # Warning: the .pbf file is 400MB
# Match with numeric coordinates (EPSG = 4326)
oe_match(c(9.1916, 45.4650), quiet = FALSE)

# Check also alternative providers
baku = oe_get(place = "Baku")

# Other examples:
oe_get("RU", match_by = "iso3166_1_alpha2", quiet = FALSE)
# The following example mimics read_sf
oe_get("Andora", stringsAsFactors = FALSE, quiet = TRUE, as_tibble = TRUE)
## End(Not run)

```

```
# Remove .pbf and .gpkg files in tempdir
oe_clean(tempdir())
```

---

oe\_get\_boundary      *Get the administrative boundary for a given place*

---

## Description

This function can be used to obtain polygon/multipolygon objects representing an administrative boundary. The objects are extracted from the multipolygons layer of a given OSM extract.

## Usage

```
oe_get_boundary(place, name = place, exact = TRUE, ...)
```

## Arguments

place	Description of the geographical area that should be matched with a <code>.osm.pbf</code> file. Can be either a length-1 character vector, an <code>sf/sfc/bbox</code> object with any CRS, or a numeric vector of coordinates with length 2. In the last case, it is assumed that the EPSG code is 4326 specified as <code>c(LON, LAT)</code> , while you can use any CRS with <code>sf/sfc/bbox</code> objects. See Details and Examples in <a href="#">oe_match()</a> .
name	A character vector of length 1 that describes the relevant area. By default, this is equal to <code>place</code> , but this parameter can be tuned to obtain more granular results starting from the same OSM extract. See examples. It must be always set when the <code>place</code> argument is specified using numeric or spatial objects.
exact	Boolean of length 1. If <code>TRUE</code> , then the function returns only those features where the field name is exactly equal to <code>name</code> . If <code>FALSE</code> , it performs a (case-sensitive) pattern matching.
...	Further arguments (e.g. <code>quiet</code> or <code>force_vectortranslate</code> ) that are passed to <code>oe_get()</code> .

## Details

The function may return an empty result when the corresponding GPKG file already exists and contains partial results. In that case, you can try running the function setting `never_skip_vectortranslate = TRUE`.

## Value

An `sf` object

**Examples**

```
## Not run:
library(sf)
my_cols = sf.colors(5, categorical = TRUE)
gabon = oe_get_boundary("Gabon", quiet = TRUE) # country
libreville = oe_get_boundary("Gabon", "Libreville", quiet = TRUE) # capital

opar = par(mar = rep(0, 4))
plot(st_geometry(st_boundary(gabon)), reset = FALSE, col = "grey")
plot(st_geometry(libreville), add = TRUE, col = my_cols[1])

# Exact match
komo = oe_get_boundary("Gabon", "Komo", quiet = TRUE)
# Pattern matching
komo_pt = oe_get_boundary("Gabon", "Komo", exact = FALSE, quiet = TRUE)
plot(st_geometry(komo), add = TRUE, col = my_cols[2])
plot(st_geometry(komo_pt), add = TRUE, col = my_cols[3:5])
par(opar)

# Get all boundaries
(oe_get_boundary("Gabon", name = "%", exact = FALSE, quiet = TRUE)[, 1:2])

# If the basic approach doesn't work, i.e.
oe_get_boundary("Leeds")

# try to consider larger regions, i.e.
oe_get_boundary("West Yorkshire", "Leeds")

## End(Not run)
```

---

oe\_get\_keys

*Return keys and (optionally) values stored in "other\_tags" column*


---

**Description**

This function returns the OSM keys and (optionally) the values stored in the other\_tags field. See Details. In both cases, the keys are sorted according to the number of occurrences, which means that the most common keys are stored first.

**Usage**

```
oe_get_keys(
  zone,
  layer = "lines",
  values = FALSE,
  which_keys = NULL,
  download_directory = oe_download_directory()
)
```

```

## Default S3 method:
oe_get_keys(
  zone,
  layer = "lines",
  values = FALSE,
  which_keys = NULL,
  download_directory = oe_download_directory()
)

## S3 method for class 'character'
oe_get_keys(
  zone,
  layer = "lines",
  values = FALSE,
  which_keys = NULL,
  download_directory = oe_download_directory()
)

## S3 method for class 'sf'
oe_get_keys(
  zone,
  layer = "lines",
  values = FALSE,
  which_keys = NULL,
  download_directory = oe_download_directory()
)

## S3 method for class 'oe_key_values_list'
print(x, n = getOption("oe_max_print_keys", 10L), ...)

```

## Arguments

zone	An sf object with an other_tags field or a character vector (of length 1) that can be linked to or pointing to a .osm.pbf or .gpkg file with an other_tags field. Character vectors are linked to .osm.pbf files using oe_find().
layer	Which layer should be read in? Typically points, lines (the default), multilinestrings, multipolygons or other_relations. If you specify an ad-hoc query using the argument query (see introductory vignette and examples), then oe_get() and oe_read() will read the layer specified in the query and ignore layer argument. See also #122.
values	Logical. If TRUE, then function returns the keys and the corresponding values, otherwise only the keys. Defaults to FALSE.
which_keys	Character vector used to subset only some keys and corresponding values. Ignored if values is FALSE. See examples.
download_directory	Path of the directory that stores the .osm.pbf files. Only relevant when zone is as a character vector that must be matched to a file via oe_find(). Ignored unless zone is a character vector.



x	object of class <code>oe_key_values_list</code>
n	Maximum number of keys (and corresponding values) to print; can be set globally by options( <code>oe_max_print_keys=...</code> ). Default value is 10.
...	Ignored.

### Details

OSM data are typically documented using several **tags**, i.e. pairs of two items, namely a key and a value. The conversion between `.osm.pbf` and `.gpkg` formats is governed by a `CONFIG` file that lists which tags must be explicitly added to the `.gpkg` file. All the other keys are automatically stored using an `other_tags` field with a syntax compatible with the PostgreSQL `HSTORE` type. See [here](#) for more details.

When the argument `values` is `TRUE`, then the function returns a named list of class `oe_key_values_list` that, for each key, summarises the corresponding values. The key-value pairs are stored using the following format: `list(key1 = c("value1", "value1", "value2", ...), key2 = c("value1", ...) ...)`. We decided to implement an ad-hoc method for printing objects of class `oe_key_values_list` using the following structure:

```
key1 = {#value1 = n1; #value2 = n2; #value3 = n3,
...} key2 = {#value1 = n1; #value2 = n2; ...} key3 = {#value1 = n1} ...
```

where `n1` denotes the number of times that `value1` is repeated, `n2` denotes the number of times that `value2` is repeated and so on. Also the values are listed according to the number of occurrences in decreasing order. By default, the function prints only the ten most common keys, but the number can be adjusted using the option `oe_max_print_keys`.

Finally, the `hstore_get_value()` function can be used inside the query argument in `oe_get()` to extract one particular tag from an existing file. Check the introductory vignette and see examples.

### Value

If the argument `values` is `FALSE` (the default), then the function returns a character vector with the names of all keys stored in the `other_tags` field. If `values` is `TRUE`, then the function returns named list which stores all keys and the corresponding values. In the latter case, the returned object has class `oe_key_values_list` and we defined an ad-hoc printing method. See `Details`.

### See Also

`oe_vectortranslate()`

### Examples

```
# Copy the ITS file to tempdir() to make sure that the examples do not
# require internet connection. You can skip the next 4 lines (and start
# directly with oe_get_keys) when running the examples locally.
```

```
its_pbf = file.path(tempdir(), "test_its-example.osm.pbf")
file.copy(
  from = system.file("its-example.osm.pbf", package = "osmextract"),
  to = its_pbf,
```

```

    overwrite = TRUE
  )

# Get keys
oe_get_keys("ITS Leeds", download_directory = tempdir())

# Get keys and values
oe_get_keys("ITS Leeds", values = TRUE, download_directory = tempdir())

# Subset some keys
oe_get_keys(
  "ITS Leeds", values = TRUE, which_keys = c("surface", "lanes"),
  download_directory = tempdir()
)

# Print all (non-NA) values for a given set of keys
res = oe_get_keys("ITS Leeds", values = TRUE, download_directory = tempdir())
res["surface"]

# Get keys from an existing sf object
its = oe_get("ITS Leeds", download_directory = tempdir())
oe_get_keys(its, values = TRUE)

# Get keys from a character vector pointing to a file (might be faster than
# reading the complete file and then filter it)
its_path = oe_get(
  "ITS Leeds", download_only = TRUE,
  download_directory = tempdir(), quiet = TRUE
)
oe_get_keys(its_path, values = TRUE)

# Add a key to an existing .gpkg file without repeating the
# vectortranslate operations
its = oe_get("ITS Leeds", download_directory = tempdir())
colnames(its)
its_extra = oe_read(
  its_path,
  query = "SELECT *, hstore_get_value(other_tags, 'oneway') AS oneway FROM lines",
  quiet = TRUE
)
colnames(its_extra)

# The following fails since there is no points layer in the .gpkg file
## Not run:
oe_get_keys(its_path, layer = "points")
## End(Not run)

# Add layer and read keys
its_path = oe_get(
  "ITS Leeds", layer = "points", download_only = TRUE,
  download_directory = tempdir(), quiet = TRUE
)
oe_get_keys(its_path, layer = "points")

```

```
# Remove .pbf and .gpkg files in tempdir
rm(its_pbf, res, its_path, its, its_extra)
oe_clean(tempdir())
```

---

oe\_get\_network      *Import transport networks used by a specific mode of transport*

---

## Description

This function is a wrapper around `oe_get()` and can be used to import a road network given a place and a mode of transport. Check the Details for a precise description of the procedures used to filter the OSM ways according to each each mode of transport.

## Usage

```
oe_get_network(place, mode = c("cycling", "driving", "walking"), ...)
```

## Arguments

place	Description of the geographical area that should be matched with a <code>.osm.pbf</code> file. Can be either a length-1 character vector, an <code>sf/sfc/bbox</code> object with any CRS, or a numeric vector of coordinates with length 2. In the last case, it is assumed that the EPSG code is 4326 specified as <code>c(LON, LAT)</code> , while you can use any CRS with <code>sf/sfc/bbox</code> objects. See Details and Examples in <code>oe_match()</code> .
mode	A character string of length one denoting the desired mode of transport. Can be abbreviated. Currently <code>cycling</code> (the default), <code>driving</code> and <code>walking</code> are supported.
...	Additional arguments passed to <code>oe_get()</code> such as <code>boundary</code> or <code>force_download</code> .

## Details

The definition of usable transport network was taken from the Python packages `osmnx` and `pyrosm` and several other documents found online, i.e. [https://wiki.openstreetmap.org/wiki/OSM\\_tags\\_for\\_routing/Access\\_restrictions](https://wiki.openstreetmap.org/wiki/OSM_tags_for_routing/Access_restrictions), <https://wiki.openstreetmap.org/wiki/Key:access>. See also the discussion in <https://github.com/ropensci/osmextract/issues/153>.

The `cycling` mode of transport (i.e. the default value for `mode` parameter) selects the OSM ways that meet the following conditions:

- The `highway` tag is not missing;
- The `highway` tag is not equal to `abandoned`, `bus_guideway`, `byway`, `construction`, `corridor`, `elevator`, `fixme`, `escalator`, `gallop`, `historic`, `no`, `planned`, `platform`, `proposed`, `raceway` or `steps`;
- The `highway` tag is not equal to `motorway`, `motorway_link`, `footway`, `bridleway` or `pedestrian` unless the tag `bicycle` is equal to `yes`, `designated`, `permissive` or `destination` (see [here](#) for more details);

- The access tag is not equal to private or no unless bicycle is equal to yes, permissive or designated (see #289);
- The bicycle tag is not equal to no, use\_sidepath, private, or restricted;
- The service tag does not contain the string private (i.e. private, private\_access and similar);

The walking mode of transport selects the OSM ways that meet the following conditions:

- The highway tag is not missing;
- The highway tag is not equal to abandoned, bus\_guideway, byway, construction, corridor, elevator, fixme, escalator, gallop, historic, no, planned, platform, proposed, raceway, motorway or motorway\_link;
- The highway tag is not equal to cycleway unless the foot tag is equal to yes;
- The access tag is not equal to private or no unless foot is equal to yes, permissive, or designated (see #289);
- The foot tag is not equal to no, use\_sidepath, private, or restricted;
- The service tag does not contain the string private (i.e. private, private\_access and similar).

The driving mode of transport selects the OSM ways that meet the following conditions:

- The highway tag is not missing;
- The highway tag is not equal to abandoned, bus\_guideway, byway, construction, corridor, elevator, fixme, escalator, gallop, historic, no, planned, platform, proposed, cycleway, pedestrian, bridleway, path, or footway;
- The access tag is not equal to private or no unless motor\_vehicle is equal to yes, permissive, or designated (see #289);
- The service tag does not contain the string private (i.e. private, private\_access and similar).

Feel free to create a new issue in the [github repo](#) if you want to suggest modifications to the current filters or propose new values for alternative modes of transport.

### Value

An sf object.

### See Also

[oe\\_get\(\)](#)

### Examples

```
# Copy the ITS file to tempdir() to make sure that the examples do not
# require internet connection. You can skip the next 4 lines (and start
# directly with oe_get_keys) when running the examples locally.
```

```
its_pbf = file.path(tempdir(), "test_its-example.osm.pbf")
```

```

file.copy(
  from = system.file("its-example.osm.pbf", package = "osmextract"),
  to = its_pbf,
  overwrite = TRUE
)

# default value returned by OSM
its = oe_get(
  "ITS Leeds", quiet = TRUE, download_directory = tempdir()
)
plot(its["highway"], lwd = 2, key.pos = 4, key.width = lcm(2.75))
# walking mode of transport
its_walking = oe_get_network(
  "ITS Leeds", mode = "walking",
  download_directory = tempdir(), quiet = TRUE
)
plot(its_walking["highway"], lwd = 2, key.pos = 4, key.width = lcm(2.75))
# driving mode of transport
its_driving = oe_get_network(
  "ITS Leeds", mode = "driving",
  download_directory = tempdir(), quiet = TRUE
)
plot(its_driving["highway"], lwd = 2, key.pos = 4, key.width = lcm(2.75))

# Remove .pbf and .gpkg files in tempdir
oe_clean(tempdir())

```

---

oe\_match

*Match input place with a url*


---

## Description

This function is used to match an input place with the URL of a .osm.pbf file (and its file-size, if present). The URLs are stored in several provider's databases. See [oe\\_providers\(\)](#) and examples.

## Usage

```

oe_match(place, ...)

## Default S3 method:
oe_match(place, ...)

## S3 method for class 'bbox'
oe_match(place, ...)

## S3 method for class 'sf'
oe_match(place, ...)

## S3 method for class 'sfc'

```

```

oe_match(place, provider = "geofabrik", level = NULL, quiet = FALSE, ...)

## S3 method for class 'numeric'
oe_match(place, provider = "geofabrik", quiet = FALSE, ...)

## S3 method for class 'character'
oe_match(
  place,
  provider = "geofabrik",
  quiet = FALSE,
  match_by = "name",
  max_string_dist = 1,
  ...
)

```

### Arguments

place	Description of the geographical area that should be matched with a <code>.osm.pbf</code> file. Can be either a length-1 character vector, an <code>sf/sfc/bbox</code> object with any CRS, or a numeric vector of coordinates with length 2. In the last case, it is assumed that the EPSG code is 4326 specified as <code>c(LON, LAT)</code> , while you can use any CRS with <code>sf/sfc/bbox</code> objects. See Details and Examples in <code>oe_match()</code> .
...	arguments passed to other methods
provider	Which provider should be used to download the data? Available providers can be browsed with <code>oe_providers()</code> . For <code>oe_get()</code> and <code>oe_match()</code> , if <code>place</code> is equal to ITS Leeds, then <code>provider</code> is internally set equal to "test". This is just for simple examples and internal tests.
level	An integer representing the desired hierarchical level in case of spatial matching. For the <code>geofabrik</code> provider, for example, 1 corresponds with continent-level datasets, 2 for countries, 3 corresponds to regions and 4 to subregions. Hence, we could approximately say that smaller administrative units correspond to bigger levels. If <code>NULL</code> , the default, the <code>oe_*</code> functions will select the highest available level. See Details and Examples in <code>oe_match()</code> .
quiet	Boolean. If <code>FALSE</code> , the function prints informative messages. Starting from <code>sf</code> version 0.9.6, if <code>quiet</code> is equal to <code>FALSE</code> , then <code>vectortranslate</code> operations will display a progress bar.
match_by	Which column of the provider's database should be used for matching the input <code>place</code> with a <code>.osm.pbf</code> file? The default is "name". Check Details and Examples in <code>oe_match()</code> to understand how this parameter works. Ignored when <code>place</code> is not a character vector since, in that case, the matching is performed through a spatial operation.
max_string_dist	Numerical value greater or equal than 0. What is the maximum distance in fuzzy matching (i.e. Approximate String Distance, see <code>adist()</code> ) between input <code>place</code> and <code>match_by</code> column that can be tolerated before testing alternative providers or looking for geographical matching with Nominatim API? This parameter is set equal to 0 if <code>match_by</code> is equal to <code>iso3166_1_alpha2</code> or <code>iso3166_2</code> . Check

Details and Examples in `oe_match()` to understand why this parameter is important. Ignored when `place` is not a character vector since, in that case, the matching is performed through a spatial operation.

## Details

If the input `place` is specified as a spatial object (either `sf` or `sfc`), then the function will return a geographical area that completely contains the object (or an error). The argument `level` (which must be specified as an integer between 1 and 4, extreme values included) is used to select between multiple geographically nested areas. We could roughly say that smaller administrative units correspond to higher levels. Check the help page of the chosen provider for more details on `level` field. By default, `level = NULL`, which means that `oe_match()` will return the area corresponding to the highest available level. If there is no geographical area at the desired level, then the function will return an error. If there are multiple areas at the same `level` intersecting the input `place`, then the function will return the area whose centroid is closest to the input `place`.

If the input `place` is specified as a character vector and there are multiple plausible matches between the input `place` and the `match_by` column, then the function will return a warning and it will select the first match. See Examples. On the other hand, if the approximate string distance between the input `place` and the best match in `match_by` column is greater than `max_string_dist`, then the function will look for exact matches (i.e. `max_string_dist = 0`) in the other supported providers. If it finds an exact match, then it will return the corresponding URL. Otherwise, if `match_by` is equal to "name", then it will try to geolocate the input `place` using the [Nominatim API](#), and then it will perform a spatial matching operation (see Examples and introductory vignette), while, if `match_by != "name"`, then it will return an error.

The fields `iso3166_1_alpha2` and `iso3166_2` are used by Geofabrik provider to perform matching operations using [ISO 3166-1 alpha-2](#) and [ISO 3166-2](#) codes. See [geofabrik\\_zones](#) for more details.

## Value

A list with two elements, named `url` and `file_size`. The first element is the URL of the `.osm.pbf` file associated with the input `place`, while the second element is the size of the file in bytes (which may be `NULL` or `NA`)

## See Also

[oe\\_providers\(\)](#) and [oe\\_match\\_pattern\(\)](#).

## Examples

```
# The simplest example:
oe_match("Italy")

# The default provider is "geofabrik", but we can change that:
oe_match("Leeds", provider = "bbbike")

# By default, the matching operations are performed through the column
# "name" in the provider's database but this can be a problem. Hence,
# you can perform the matching operations using other columns:
oe_match("RU", match_by = "iso3166_1_alpha2")
# Run oe_providers() for reading a short description of all providers and
```

```

# check the help pages of the corresponding databases to learn which fields
# are present.

# You can always increase the max_string_dist argument, but it can be
# dangerous:
oe_match("London", max_string_dist = 3, quiet = FALSE)

# Match the input zone using an sfc object:
milan_duomo = sf::st_sfc(sf::st_point(c(1514924, 5034552)), crs = 3003)
oe_match(milan_duomo, quiet = FALSE)
leeds = sf::st_sfc(sf::st_point(c(430147.8, 433551.5)), crs = 27700)
oe_match(leeds, provider = "bbbike")

# If you specify more than one sfg object, then oe_match will select the OSM
# extract that covers all areas
milan_leeds = sf::st_sfc(
  sf::st_point(c(9.190544, 45.46416)), # Milan
  sf::st_point(c(-1.543789, 53.7974)), # Leeds
  crs = 4326
)
oe_match(milan_leeds)

# Match the input zone using a numeric vector of coordinates
# (in which case crs = 4326 is assumed)
oe_match(c(9.1916, 45.4650)) # Milan, Duomo using CRS = 4326

# The following returns a warning since Berin is matched both
# with Benin and Berlin
oe_match("Berin", quiet = FALSE)

# If the input place does not match any zone in the chosen provider, then the
# function will test the other providers:
oe_match("Leeds")

# If the input place cannot be exactly matched with any zone in any provider,
# then the function will try to geolocate the input and then it will perform a
# spatial match:
## Not run:
oe_match("Milan")
## End(Not run)

# The level parameter can be used to select smaller or bigger geographical
# areas during spatial matching
yak = c(-120.51084, 46.60156)
## Not run:
oe_match(yak, level = 3) # error
oe_match(yak, level = 2) # by default, level is equal to the maximum value
oe_match(yak, level = 1)
## End(Not run)

```



**Description**

This function is used to explore all provider's databases and look for matches. This function can be useful in combination with `oe_match()` and `oe_get()` for an exploratory analysis and an easy match. See Examples.

**Usage**

```
oe_match_pattern(pattern, ...)

## S3 method for class 'numeric'
oe_match_pattern(pattern, full_row = FALSE, ...)

## S3 method for class 'sf'
oe_match_pattern(pattern, full_row = FALSE, ...)

## S3 method for class 'bbox'
oe_match_pattern(pattern, full_row = FALSE, ...)

## S3 method for class 'sfc'
oe_match_pattern(pattern, full_row = FALSE, ...)

## S3 method for class 'character'
oe_match_pattern(pattern, match_by = "name", full_row = FALSE, ...)
```

**Arguments**

pattern	Description of the pattern. Can be either a length-1 character vector, an sf/sfc/bbox object, or a numeric vector of coordinates with length 2. In the last case, it is assumed that the EPSG code is 4326 specified as c(LON, LAT), while you can use any CRS with sf/sfc/bbox objects.
...	arguments passed to other methods
full_row	Boolean. Return all columns for the matching rows? FALSE by default.
match_by	Name of the column in the provider's database that will be used to find the match in case of character input. In all the other cases, the match is performed using a spatial overlay operation and the output returns the values stored in the name column (or even the full sf object when full_row is TRUE).

**Value**

A list of character vectors or sf objects (according to the value of the parameter full\_row). If no OSM zone can be matched with the input string, then the function returns an empty list.

**Examples**

```
oe_match_pattern("Yorkshire")

res = oe_match_pattern("Yorkshire", full_row = TRUE)
lapply(res, function(x) sf::st_drop_geometry(x)[, 1:3])
```

```
oe_match_pattern(c(9, 45)) # long/lat for Milan, Italy
```

---

oe_providers	<i>Summary of available providers</i>
--------------	---------------------------------------

---

### Description

This function is used to display a short summary of the major characteristics of the databases associated to all available providers.

### Usage

```
oe_providers(quiet = FALSE)
```

### Arguments

quiet	Boolean. If FALSE, the function prints informative messages. Starting from <code>sf</code> version <b>0.9.6</b> , if quiet is equal to FALSE, then <code>vectortranslate</code> operations will display a progress bar.
-------	---

### Value

A `data.frame` with 4 columns representing the name of each available provider, the name of the corresponding database and the number of features and fields.

### Examples

```
oe_providers()
```

---

oe_read	<i>Read a .pbf or .gpkg object from file or url</i>
---------	---

---

### Description

This function is used to read a `.pbf` or `.gpkg` object from file or URL. It is a wrapper around `oe_download()`, `oe_vectortranslate()`, and `sf::st_read()`, creating an easy way to download, convert, and read a `.pbf` or `.gpkg` file. Check the introductory vignette and the help pages of the wrapped function for more details.

**Usage**

```

oe_read(
  file_path,
  layer = "lines",
  ...,
  provider = NULL,
  download_directory = oe_download_directory(),
  file_size = NULL,
  force_download = FALSE,
  max_file_size = 5e+08,
  download_only = FALSE,
  skip_vectortranslate = FALSE,
  vectortranslate_options = NULL,
  osmconf_ini = NULL,
  extra_tags = NULL,
  force_vectortranslate = FALSE,
  never_skip_vectortranslate = FALSE,
  boundary = NULL,
  boundary_type = c("spat", "clipsrc"),
  quiet = FALSE
)

```

**Arguments**

<code>file_path</code>	A URL or the path to a .pbf or .gpkg file. If a URL, then it must be specified using HTTP/HTTPS protocol.
<code>layer</code>	Which layer should be read in? Typically points, lines (the default), multilinestrings, multipolygons or other_relations. If you specify an ad-hoc query using the argument <code>query</code> (see introductory vignette and examples), then <code>oe_get()</code> and <code>oe_read()</code> will read the layer specified in the query and ignore <code>layer</code> argument. See also <a href="#">#122</a> .
<code>...</code>	(Named) arguments that will be passed to <code>sf::st_read()</code> , like <code>query</code> , <code>wkt_filter</code> or <code>stringsAsFactors</code> . Check the introductory vignette to understand how to create your own (SQL-like) queries.
<code>provider</code>	Which provider should be used to download the data? Available providers can be browsed with <code>oe_providers()</code> . For <code>oe_get()</code> and <code>oe_match()</code> , if <code>place</code> is equal to ITS Leeds, then <code>provider</code> is internally set equal to "test". This is just for simple examples and internal tests.
<code>download_directory</code>	Directory to store the file containing OSM data?.
<code>file_size</code>	How big is the file? Optional. NA by default. If it's bigger than <code>max_file_size</code> and the function is run in interactive mode, then an interactive menu is displayed, asking for permission to download the file.
<code>force_download</code>	Should the .osm.pbf file be updated even if it has already been downloaded? FALSE by default. This parameter is used to update old .osm.pbf files.
<code>max_file_size</code>	The maximum file size to download without asking in interactive mode. Default: 5e+8, half a gigabyte.

download_only	Boolean. If TRUE, then the function only returns the path where the matched file is stored, instead of reading it. FALSE by default.
skip_vectortranslate	Boolean. If TRUE, then the function skips all vectortranslate operations and it reads (or simply returns the path) of the .osm.pbf file. FALSE by default.
vectortranslate_options	Options passed to the <code>sf::gdal_utils()</code> argument options. Set by default. Check details in the introductory vignette and the help page of <code>oe_vectortranslate()</code> .
osmconf_ini	The configuration file. See documentation at <a href="http://gdal.org">gdal.org</a> . Check details in the introductory vignette and the help page of <code>oe_vectortranslate()</code> . Set by default.
extra_tags	Which additional columns, corresponding to OSM tags, should be in the resulting dataset? NULL by default. Check the introductory vignette and the help pages of <code>oe_vectortranslate()</code> and <code>oe_get_keys()</code> . Ignored when <code>osmconf_ini</code> is not NULL.
force_vectortranslate	Boolean. Force the original .pbf file to be translated into a .gpkg file, even if a .gpkg with the same name already exists? FALSE by default. If tags in <code>extra_tags</code> match data in previously translated .gpkg files no translation occurs (see <a href="#">#173</a> for details). Check the introductory vignette and the help page of <code>oe_vectortranslate()</code> .
never_skip_vectortranslate	Boolean. This is used in case the user passed its own .ini file or vectortranslate options (since, in those case, it's too difficult to determine if an existing .gpkg file was generated following the same options.)
boundary	An <code>sf/sfc/bbox</code> object that will be used to create a spatial filter during the vectortranslate operations. The type of filter can be chosen using the argument <code>boundary_type</code> .
boundary_type	A character vector of length 1 specifying the type of spatial filter. The spat filter selects only those features that intersect a given area, while <code>clipsrc</code> also clips the geometries. Check the examples and also <a href="#">here</a> for more details.
quiet	Boolean. If FALSE, the function prints informative messages. Starting from <code>sf</code> version <a href="#">0.9.6</a> , if <code>quiet</code> is equal to FALSE, then <code>vectortranslate</code> operations will display a progress bar.

## Details

The arguments `provider`, `download_directory`, `file_size`, `force_download`, and `max_file_size` are ignored if `file_path` points to an existing .pbf or .gpkg file.

Please note that you cannot add any field to an existing .gpkg file using the argument `extra_tags` without rerunning the `vectortranslate` process on the corresponding .pbf file. On the other hand, you can extract some of the tags in `other_tags` field as new columns. See examples and `oe_get_keys()` for more details.

## Value

An `sf` object or, when `download_only` argument equals TRUE, a character vector.

**Examples**

```

# Read an existing .pbf file. First we need to copy a .pbf file into a
# temporary directory
its_pbf = file.path(tempdir(), "test_its-example.osm.pbf")
file.copy(
  from = system.file("its-example.osm.pbf", package = "osmextract"),
  to = its_pbf
)
oe_read(its_pbf)

# Read a new layer
oe_read(its_pbf, layer = "points")

# The following example shows how to add new tags
names(oe_read(its_pbf, extra_tags = c("oneway", "ref"), quiet = TRUE))

# Read an existing .gpkg file. This file was created internally by oe_read().
its_gpkg = file.path(tempdir(), "test_its-example.gpkg")
oe_read(its_gpkg)

# You cannot add any new layer to an existing .gpkg file but you can extract
# some of the tags in other_tags. Check oe_get_keys() for more details.
names(oe_read(its_gpkg, extra_tags = c("maxspeed"))) # doesn't work
# Instead, use the query argument
names(oe_read(
  its_gpkg,
  quiet = TRUE,
  query =
    "SELECT *,
     hstore_get_value(other_tags, 'maxspeed') AS maxspeed
    FROM lines
    ")
))

# Read from a URL
my_url = "https://github.com/ropensci/osmextract/raw/master/inst/its-example.osm.pbf"
# Please note that if you read from a URL which is not linked to one of the
# supported providers, you need to specify the provider parameter:
## Not run:
oe_read(my_url, provider = "test", quiet = FALSE)
## End(Not run)

# Remove .pbf and .gpkg files in tempdir
oe_clean(tempdir())

```

**Description**

This (only internal and experimental) function provides a simple interface to the **nominatim** service for finding the geographical location of place names.

**Usage**

```
oe_search(
  place,
  base_url = "https://nominatim.openstreetmap.org",
  destfile = tempfile(fileext = ".geojson"),
  ...
)
```

**Arguments**

place	Text string containing the name of a place the location of which is to be found, such as "Leeds" or "Milan".
base_url	The URL of the nominatim server to use. The main open server hosted by OpenStreetMap is the default.
destfile	The name of the destination file where the output of the search query, a .geojson file, should be saved.
...	Extra arguments that are passed to <code>sf::st_read</code> .

**Value**

An `sf` object corresponding to the input place. The `sf` object is read by `sf::st_read()` and it is based on a geojson file returned by Nominatim API.

---

oe_update	<i>Update all the .osm.pbf files saved in a directory</i>
-----------	---

---

**Description**

This function is used to re-download all .osm.pbf files stored in `download_directory` that were firstly downloaded through `oe_get()`. See Details.

**Usage**

```
oe_update(
  download_directory = oe_download_directory(),
  quiet = FALSE,
  delete_gpkg = TRUE,
  max_file_size = 5e+08,
  ...
)
```

**Arguments**

download_directory	Character string of the path of the directory where the .osm.pbf files are saved.
quiet	Boolean. If FALSE the function prints informative messages. See Details.
delete_gpkg	Boolean. if TRUE the function deletes the old .gpkg files. We added this parameter to minimize the probability of accidentally reading-in old and not-synchronized .gpkg files. See Details. Defaults to TRUE.
max_file_size	The maximum file size to download without asking in interactive mode. Default: 5e+8, half a gigabyte.
...	Additional parameter that will be passed to <code>oe_get()</code> (such as <code>stringsAsFactors</code> or <code>query</code> ).

**Details**

This function is used to re-download .osm.pbf files that are stored in a directory (specified by `download_directory` param) and that were firstly downloaded through `oe_get()`. The name of the files must begin with the name of one of the supported providers (see `oe_providers()`) and it must end with .osm.pbf. All other files in the directory that do not match this format are ignored.

The process for re-downloading the .osm.pbf files is performed using the function `oe_get()`. The appropriate provider is determined by looking at the first word in the path of the .osm.pbf file. The place is determined by looking at the second word in the file path and the matching is performed through the `id` column in the provider's database. So, for example, the path `geofabrik_italy-latest-update.osm.pbf` will be matched with the provider "geofabrik" and the geographical zone `italy` through the column `id` in `geofabrik_zones`.

The parameter `delete_gpkg` is used to delete all .gpkg files in `download_directory`. We decided to set its default value to TRUE to minimize the possibility of reading-in old and non-synchronized .gpkg files. If you set `delete_gpkg = FALSE`, then you need to manually reconvert all files using `oe_get()` or `oe_vectortranslate()`.

If you set the parameter `quiet` to FALSE, then the function will print some useful messages regarding the characteristics of the files before and after updating them. More precisely, it will print the output of the columns `size`, `mtime` and `ctime` from `file.info()`. Please note that the meaning of `mtime` and `ctime` depends on the OS and the file system. Check `file.info()`.

**Value**

The path(s) of the .osm.pbf file(s) that were updated.

**Examples**

```
## Not run:
# Set up a fake directory with .pbf and .gpkg files
fake_dir = tempdir()
# Fill the directory
oe_get("Andorra", download_directory = fake_dir, download_only = TRUE)
# Check the directory
list.files(fake_dir, pattern = "gpkg|pbf")
# Update all .pbf files and delete all .gpkg files
```

```
oe_update(fake_dir, quiet = TRUE)
list.files(fake_dir, pattern = "gpkg|pbf")
## End(Not run)
```

---

oe\_vectortranslate      *Translate a .osm.pbf file into .gpkg format*

---

## Description

This function is used to translate a .osm.pbf file into .gpkg format. The conversion is performed using `ogr2ogr` via the `vectortranslate` utility in `sf::gdal_utils()`. It was created following [the suggestions](#) of the maintainers of GDAL. See [Details](#) and [Examples](#) to understand the basic usage, and check the introductory vignette for more complex use-cases.

## Usage

```
oe_vectortranslate(
  file_path,
  layer = "lines",
  vectortranslate_options = NULL,
  osmconf_ini = NULL,
  extra_tags = NULL,
  force_vectortranslate = FALSE,
  never_skip_vectortranslate = FALSE,
  boundary = NULL,
  boundary_type = c("spat", "clipsrc"),
  quiet = FALSE
)
```

## Arguments

<code>file_path</code>	Character string representing the path of the input .pbf or .osm.pbf file.
<code>layer</code>	Which layer should be read in? Typically points, lines (the default), multilinestrings, multipolygons or other_relations. If you specify an ad-hoc query using the argument query (see introductory vignette and examples), then <code>oe_get()</code> and <code>oe_read()</code> will read the layer specified in the query and ignore layer argument. See also <a href="#">#122</a> .
<code>vectortranslate_options</code>	Options passed to the <code>sf::gdal_utils()</code> argument options. Set by default. Check details in the introductory vignette and the help page of <code>oe_vectortranslate()</code> .
<code>osmconf_ini</code>	The configuration file. See documentation at <a href="http://gdal.org">gdal.org</a> . Check details in the introductory vignette and the help page of <code>oe_vectortranslate()</code> . Set by default.
<code>extra_tags</code>	Which additional columns, corresponding to OSM tags, should be in the resulting dataset? NULL by default. Check the introductory vignette and the help pages of <code>oe_vectortranslate()</code> and <code>oe_get_keys()</code> . Ignored when <code>osmconf_ini</code> is not NULL.



<code>force_vectortranslate</code>	Boolean. Force the original <code>.pbf</code> file to be translated into a <code>.gpkg</code> file, even if a <code>.gpkg</code> with the same name already exists? <code>FALSE</code> by default. If tags in <code>extra_tags</code> match data in previously translated <code>.gpkg</code> files no translation occurs (see <a href="#">#173</a> for details). Check the introductory vignette and the help page of <code>oe_vectortranslate()</code> .
<code>never_skip_vectortranslate</code>	Boolean. This is used in case the user passed its own <code>.ini</code> file or <code>vectortranslate</code> options (since, in those case, it's too difficult to determine if an existing <code>.gpkg</code> file was generated following the same options.)
<code>boundary</code>	An <code>sf/sfc/bbox</code> object that will be used to create a spatial filter during the <code>vectortranslate</code> operations. The type of filter can be chosen using the argument <code>boundary_type</code> .
<code>boundary_type</code>	A character vector of length 1 specifying the type of spatial filter. The spat filter selects only those features that intersect a given area, while <code>clipsrc</code> also clips the geometries. Check the examples and also <a href="#">here</a> for more details.
<code>quiet</code>	Boolean. If <code>FALSE</code> , the function prints informative messages. Starting from <code>sf</code> version <a href="#">0.9.6</a> , if <code>quiet</code> is equal to <code>FALSE</code> , then <code>vectortranslate</code> operations will display a progress bar.

## Details

The new `.gpkg` file is created in the same directory as the input `.osm.pbf` file. The translation process is performed using the `vectortranslate` utility in `sf::gdal_utils()`. This operation can be customized in several ways modifying the parameters `layer`, `extra_tags`, `osmconf_ini`, `vectortranslate_options`, `boundary` and `boundary_type`.

The `.osm.pbf` files processed by GDAL are usually categorized into 5 layers, named `points`, `lines`, `multilinestrings`, `multipolygons` and `other_relations`. Check the first paragraphs [here](#) for more details. This function can convert only one layer at a time, and the parameter `layer` is used to specify which layer of the `.osm.pbf` file should be converted. Several layers with different names can be stored in the same `.gpkg` file. By default, the function will convert the `lines` layer (which is the most common one according to our experience).

The arguments `osmconf_ini` and `extra_tags` are used to modify how GDAL reads and processes a `.osm.pbf` file. More precisely, several operations that GDAL performs on the input `.osm.pbf` file are governed by a `CONFIG` file, that can be checked at the following [link](#). The basic components of OSM data are called *elements* and they are divided into *nodes*, *ways* or *relations*, so, for example, the code at line 7 of that file is used to determine which *ways* are assumed to be polygons (according to the simple-feature definition of polygon) if they are closed. Moreover, OSM data is usually described using several *tags*, i.e pairs of two items: a key and a value. The code at lines 33, 53, 85, 103, and 121 is used to determine, for each layer, which tags should be explicitly reported as fields (while all the other tags are stored in the `other_tags` column). The parameter `extra_tags` is used to determine which extra tags (i.e. key/value pairs) should be added to the `.gpkg` file (other than the default ones).

By default, the `vectortranslate` operations are skipped if the function detects a file having the same path as the input file, `.gpkg` extension, a layer with the same name as the parameter `layer` and all `extra_tags`. In that case the function will simply return the path of the `.gpkg` file. This behaviour can be overwritten setting `force_vectortranslate = TRUE`. The `vectortranslate` operations

are never skipped if `osmconf_ini`, `vectortranslate_options`, `boundary` or `boundary_type` arguments are not NULL.

The parameter `osmconf_ini` is used to pass your own CONFIG file in case you need more control over the GDAL operations. Check the package introductory vignette for an example. If `osmconf_ini` is equal to NULL (the default value), then the function uses the standard `osmconf.ini` file defined by GDAL (but for the extra tags).

The parameter `vectortranslate_options` is used to control the options that are passed to `ogr2ogr` via `sf::gdal_utils()` when converting between `.osm.pbf` and `.gpkg` formats. `ogr2ogr` can perform various operations during the conversion process, such as spatial filters or SQL queries. These operations can be tuned using the `vectortranslate_options` argument. If NULL (the default value), then `vectortranslate_options` is set equal to

```
c("-f", "GPKG", "-overwrite", "-oo", paste0("CONFIG_FILE=", osmconf_ini), "-lco", "GEOMETRY_NAME=geometry", layer).
```

Explanation:

- `"-f", "GPKG"` says that the output format is GPKG;
- `"-overwrite"` is used to delete an existing layer and recreate it empty;
- `"-oo", paste0("CONFIG_FILE=", osmconf_ini)` is used to set the **Open Options** for the `.osm.pbf` file and change the CONFIG file (in case the user asks for any extra tag or a totally different CONFIG file);
- `"-lco", "GEOMETRY_NAME=geometry"` is used to change the **layer creation options** for the `.gpkg` file and modify the name of the geometry column;
- `layer` indicates which layer should be converted.

If `vectortranslate_options` is not NULL, then the options `c("-f", "GPKG", "-overwrite", "-oo", "CONFIG_FILE=", path-to-config-file, "-lco", "GEOMETRY_NAME=geometry", layer)` are always appended unless the user explicitly sets different default parameters for the arguments `-f`, `-oo`, `-lco`, and `layer`.

The arguments `boundary` and `boundary_type` can be used to set up a spatial filter during the vector-translate operations (and speed up the process) using an `sf` or `sfc` object (POLYGON or MULTIPOLYGON). The default arguments create a rectangular spatial filter which selects all features that intersect the area. Setting `boundary_type = "clipsrc"` clips the geometries. In both cases, the appropriate options are automatically added to the `vectortranslate_options` (unless a user explicitly sets different default options). Check Examples in `oe_get()` and the introductory vignette.

See also the help page of `sf::gdal_utils()` and `ogr2ogr` for more examples and extensive documentation on all available options that can be tuned during the `vectortranslate` process.

## Value

Character string representing the path of the `.gpkg` file.

## See Also

[oe\\_get\\_keys\(\)](#)

**Examples**

```

# First we need to match an input zone with a .osm.pbf file
(its_match = oe_match("ITS Leeds"))

# Copy ITS file to tempdir so that the examples do not require internet
# connection. You can skip the next 3 lines (and start directly with
# oe_download()) when running the examples locally.

file.copy(
  from = system.file("its-example.osm.pbf", package = "osmextract"),
  to = file.path(tempdir(), "test_its-example.osm.pbf"),
  overwrite = TRUE
)

# The we can download the .osm.pbf file (if it was not already downloaded)
its_pbf = oe_download(
  file_url = its_match$url,
  file_size = its_match$file_size,
  download_directory = tempdir(),
  provider = "test"
)

# Check that the file was downloaded
list.files(tempdir(), pattern = "pbf|gpkg")

# Convert to gpkg format
its_gpkg = oe_vectortranslate(its_pbf)

# Now there is an extra .gpkg file
list.files(tempdir(), pattern = "pbf|gpkg")

# Check the layers of the .gpkg file
sf::st_layers(its_gpkg, do_count = TRUE)

# Add points layer
its_gpkg = oe_vectortranslate(its_pbf, layer = "points")
sf::st_layers(its_gpkg, do_count = TRUE)

# Add extra tags to the lines layer
names(sf::st_read(its_gpkg, layer = "lines", quiet = TRUE))
its_gpkg = oe_vectortranslate(
  its_pbf,
  extra_tags = c("oneway", "maxspeed")
)
names(sf::st_read(its_gpkg, layer = "lines", quiet = TRUE))

# Adjust vectortranslate options and convert only 10 features
# for the lines layer
oe_vectortranslate(
  its_pbf,
  vectortranslate_options = c("-limit", 10)
)

```

```
sf::st_layers(its_gpkg, do_count = TRUE)

# Remove .pbf and .gpkg files in tempdir
oe_clean(tempdir())
```

---

openstreetmap\_fr\_zones

*An sf object of geographical zones taken from download.openstreetmap.fr*

---

### Description

An sf object containing the URLs, names, and file-sizes of the OSM extracts stored at <http://download.openstreetmap.fr/>.

### Usage

```
openstreetmap_fr_zones
```

### Format

An sf object with 1187 rows and 7 columns:

**id** A unique ID for each area. It is used by `oe_update()`.

**name** The, usually English, long-form name of the city.

**parent** The identifier of the next larger excerpts that contains this one, if present.

**level** An integer code between 1 and 4. Check <http://download.openstreetmap.fr/polygons/> to understand the hierarchical structure of the zones. 1L correspond to the biggest areas. This is used only for matching operations in case of spatial input.

**pbf** Link to the latest .osm.pbf file for this region.

**pbf\_file\_size** Size of the pbf file in bytes.

**geometry** The sfg for that geographical region, rectangular. See also `oe_get_boundary()` to extract the proper geographical boundaries.

### Source

<http://download.openstreetmap.fr/>

### See Also

Other provider's-database: [bbbike\\_zones](#), [geofabrik\\_zones](#)

---

read_poly	<i>Read a .poly file.</i>
-----------	---------------------------

---

## Description

Read a .poly file.

## Usage

```
read_poly(input, crs = "OGC:CRS84", ...)
```

## Arguments

input	Character vector representing a polygon object saved using the .poly format. Can be also a path to a file or a URL pointing to a valid .poly file.
crs	The Coordinate Reference System (CRS) of the input polygon.
...	Further arguments passed to readLines() (which is the function used to read external .poly files).

## Details

The Polygon Filter File Format (.poly) is defined [here](#). The code behind the function was inspired by the `parse_poly` function defined [here](#).

**Geofabrik** stores the .poly files used to generate their extracts. Furthermore, a nice collection of exact-border poly files created from cities with an OSM Relation ID is available in this git repository on github: <https://github.com/jameschevalier/cities>.

The default value for the `crs` argument is "OGC:CRS84" instead of "4326" or "EPSG:4326" since, by definition, the coordinates are provided as "longitude, latitude" (but these differences should be relevant only when `sf::st_axis_order()` is TRUE).

## Value

A `sfc_MULTIPOLYGON/sfc` object.

## Examples

```
toy_poly <- c(
  "test_poly",
  "first_area",
  "0 0",
  "0 1",
  "1 1",
  "1 0",
  "0 0",
  "END",
  "END"
)
```

```
(out <- read_poly(toy_poly))
plot(out)

## Not run:
italy_poly <- "https://download.geofabrik.de/europe/italy.poly"
plot(read_poly(italy_poly))
## End(Not run)
```

---

test_zones	<i>An sf object of geographical zones taken from download.openstreetmap.fr</i>
------------	--

---

### Description

This object represent a minimal provider's database and it should be used only for examples and tests.

### Usage

```
test_zones
```

### Format

An object of class `sf` (inherits from `data.frame`) with 2 rows and 7 columns.

# Index

- \* **datasets**
  - bbbike\_zones, 2
  - geofabrik\_zones, 3
  - openstreetmap\_fr\_zones, 36
  - test\_zones, 38
- \* **provider's-database**
  - bbbike\_zones, 2
  - geofabrik\_zones, 3
  - openstreetmap\_fr\_zones, 36
- adist(), 10, 22
- bbbike\_zones, 2, 4, 36
- download.file(), 6
- file.info(), 31
- geofabrik\_zones, 2, 3, 3, 23, 36
- oe\_clean, 4
- oe\_download, 5
- oe\_download(), 12, 26
- oe\_download\_directory, 6
- oe\_download\_directory(), 7
- oe\_find, 7
- oe\_get, 9
- oe\_get(), 7, 8, 10, 16, 19, 20, 22, 25, 27, 30–32
- oe\_get\_boundary, 14
- oe\_get\_keys, 15
- oe\_get\_keys(), 11, 28, 32, 34
- oe\_get\_network, 19
- oe\_match, 21
- oe\_match(), 7–10, 12, 14, 19, 22, 23, 25, 27
- oe\_match\_pattern, 24
- oe\_match\_pattern(), 23
- oe\_providers, 26
- oe\_providers(), 7, 10, 21–23, 27, 31
- oe\_read, 26
- oe\_read(), 9, 10, 12, 16, 27, 32
- oe\_search, 29
- oe\_update, 30
- oe\_vectortranslate, 32
- oe\_vectortranslate(), 11, 12, 26, 28, 31–33
- openstreetmap\_fr\_zones, 3, 4, 36
- print.oe\_key\_values\_list(oe\_get\_keys), 15
- read\_poly, 37
- sf::gdal\_utils(), 11, 28, 32–34
- sf::st\_read(), 10, 26, 27
- test\_zones, 38