

Package ‘ffm’

January 30, 2026

Type Package

Title Download Official Spatial Data from Germany

Version 0.1.2

Description Provides quick and easy access to official spatial data from Germany’s Federal Agency for Cartography and Geodesy (BKG) <<https://gdz.bkg.bund.de/>>. Interfaces various web feature services (WFS) and download servers. Allows retrieval, caching and filtering with a wide range of open geodata products, including administrative or non-administrative boundaries, land cover, elevation models, geographic names, and points of interest covering Germany. Can be particularly useful for linking regional statistics to their spatial representations and streamlining workflows that involve spatial data of Germany.

License MIT + file LICENSE

URL <https://github.com/jslth/ffm/>, <https://jslth.github.io/ffm/>

BugReports <https://github.com/jslth/ffm/issues>

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Depends R (>= 3.5)

Imports cli, rlang, zip, httr2, xml2, sf

Suggests tibble, terra, arrow, ggplot2, knitr, rmarkdown, testthat (>= 3.2.2)

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Jonas Lieth [aut, cre, cph] (ORCID: <<https://orcid.org/0000-0002-3451-3176>>)

Maintainer Jonas Lieth <jslth@outlook.com>

Repository CRAN

Date/Publication 2026-01-30 20:32:07 UTC

Contents

admin_data	2
bkg_admin	3
bkg_admin_hierarchy	8
bkg_agr	10
bkg_airports	11
bkg_area_codes	14
bkg_authorities	16
bkg_clc	18
bkg_crossings	20
bkg_dem	22
bkg_dlm	23
bkg_geonames	26
bkg_grid	30
bkg_heliports	32
bkg_kfz	35
bkg_kilometrage	38
bkg_nuts	40
bkg_quasigeoid	42
bkg_ror	43
bkg_seaports	46
bkg_stations	48
bkg_trauma_centers	51
bkg_wcs	54
bkg_wfs	55
nuts_data	57
wfs_filter	58

Index	61
--------------	-----------

admin_data	<i>German administrative boundaries</i>
------------	---

Description

Three `sf` dataframes containing all geometries of German districts, federal states, and the country, respectively. The reference year is 2023.

Usage

`bkg_krs`

`bkg_states`

`bkg_germany`

Format

For the dataframe format, see [bkg_admin](#).

An object of class sf (inherits from tbl_df, tbl, data.frame) with 25 rows and 25 columns.

An object of class sf (inherits from tbl_df, tbl, data.frame) with 7 rows and 25 columns.

Source

© BKG (2025) dl-de/by-2-0, data sources: https://sgx.geodatenzentrum.de/web_public/gdz/datenquellen/Datenquellen_vg_nuts.pdf

See Also

[bkg_admin](#)

Other datasets: [nuts_data](#)

Examples

```
bkg_krs
```

bkg_admin	<i>Administrative areas</i>
-----------	-----------------------------

Description

Retrieve polygon geometries of administrative areas in Germany. All administrative levels are supported at different spatial resolutions.

- `bkg_admin` interfaces a WFS that allows prefiltering but provides no historical data and allows a maximum scale of 1:250,000.
- `bkg_admin_archive` allows access to historical data but has no prefiltering.
- `bkg_admin_highres` (`vg25`) allows access to high-resolution data going as low as 1:25,000 but allows no prefiltering.

These functions interface the `vg*` products of the BKG.

Usage

```
bkg_admin(  
  ...,  
  level = "krs",  
  scale = c("250", "1000", "2500", "5000"),  
  key_date = c("0101", "1231"),  
  bbox = NULL,  
  poly = NULL,  
  predicate = "intersects",  
  filter = NULL,
```

```

    epsg = 3035,
    properties = NULL,
    max = NULL
)

bkg_admin_archive(
  level = "krs",
  scale = c("250", "1000", "2500", "5000"),
  key_date = c("0101", "1231"),
  year = "latest",
  timeout = 120,
  update_cache = FALSE
)

bkg_admin_highres(
  level = "krs",
  year = "latest",
  layer = NULL,
  timeout = 600,
  update_cache = FALSE
)

```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

level Administrative level to download. Must be one of "sta" (Germany), "lan" (federal states), "rbz" (governmental districts), "krs" (districts), "vwg" (administrative associations), "gem" (municipalities), "li" (boundary lines), or "pk" (municipality centroids). Defaults to districts.

scale	Scale of the geometries. Can be "250" (1:250,000), "1000" (1:1,000,000), "2500" (1:2,500,000) or "5000" (1:5,000,000). If "250", population data is included in the output. Defaults to "250".
key_date	For resolution <code>%in% c("250", "5000")</code> , specifies the key date from which to download administrative data. Can be either "0101" (January 1) or "1231" (December 31). The latter is able to georeference statistical data while the first integrates changes made in the new year. If "1231", population data is attached, otherwise not. Note that population data is not available at all scales (usually 250 and 1000). Defaults to "0101".
bbox	An sf geometry or a boundary box vector of the format <code>c(xmin, ymin, xmax, ymax)</code> . Used as a geometric filter to include only those geometries that relate to bbox according to the predicate specified in predicate. If an sf geometry is provided, coordinates are automatically transformed to ESPG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
poly	An sf geometry. Used as a geometric filter to include only those geometries that relate to poly according to the predicate specified in predicate. Coordinates are automatically transformed to ESPG:25832 (the default CRS).
predicate	A spatial predicate that is used to relate the output geometries with the object specified in bbox or poly. For example, if predicate = "within", and bbox is specified, returns only those geometries that lie within bbox. Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within", "contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
filter	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through <code>...</code> . Use this argument to construct more complex filters. Defaults to NULL.
epsg	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(...)\$epsg</code> on a spatial object that you are working with. Defaults to 3035.
properties	Vector of columns to include in the output.
max	Maximum number of results to return.
year	Version year of the dataset. You can use <code>latest</code> to retrieve the latest dataset version available on the BKG's geodata center. Older versions can be browsed using the archive .
timeout	Timeout value for the data download passed to <code>req_timeout</code> . Adjust this if your internet connection is slow or you are downloading larger datasets.
update_cache	By default, downloaded files are cached in the <code>tempdir()</code> directory of R. When downloading the same data again, the data is not downloaded but instead taken from the cache. Sometimes this can be not the desired behavior. If you want to overwrite the cache, pass <code>TRUE</code> . Defaults to <code>FALSE</code> , i.e. always adopt the cache if possible.
layer	The <code>vg25</code> product used in <code>bkg_admin_highres</code> contains a couple of metadata files. You can set a layer name to read these files, otherwise the main file is read.

Value

An sf dataframe with multipolygon geometries and different columns depending on the geometry type. Areal geometries generally have the following columns:

- objid: Unique object identifier
- beginn: Creation of the object in the DLM
- ade: Integer representing the administrative level. Can be one of
 - 1: Germany
 - 2: Federal state
 - 3: Governmental district
 - 4: District
 - 5: Administrative association
 - 6: Municipality
- gf: Integer representing the geofactor; whether an area is "structured" or not. Land is structured if it is part of a state or other administrative unit but is not further divided into administrative units. Can be one of
 - 1: Unstructured, waterbody
 - 2: Structured, waterbody
 - 3: Unstructured, land
 - 4: Structured, land
- bsg: Special areas, can be 1 (Germany) or 9 (Lake Constance)
- ars: Territorial code (Amtlicher Regionalschlüssel). The ARS is structured hierarchically as follows:
 - Position 1-2: Federal state
 - Position 3: Government region
 - Position 4-5: District
 - Position 6-9: Administrative association
 - Position 10-12: Municipality
- ags: Official municipality key (Amtlicher Gemeindeschlüssel). Related to the ARS but shortened to omit position 6 to 9. Structured as follows:
 - Position 1-2: Federal state
 - Position 3: Government region
 - Position 4-5: District
 - Position 6-8: Municipality
- sdv_ars: ARS of the seat of administration
- gen: Geographical name
- bez: Label of the administrative unit
- ibz: Identifier of the label
- bem: Comment on the label
- nbd: Formation of the geographical name. Can be "ja" if the label is part of the name or "nein" otherwise.

- nuts: NUTS identifier based on the Eurostat regional classification
- ars_0: ARS identifier with trailing zeroes
- ags_0: AGS identifier with trailing zeroes
- wsk: Legally relevant date for the effectiveness of administrative changes
- sn_l: Federal state component of the ARS
- sn_r: Governmental district component of the ARS
- sn_k: District component of the ARS
- sn_v1: First part of the administrative association component of the ARS
- sn_v2: Second part of the administrative association component of the ARS
- sn_g: Municipality component of the ARS
- fk_3: Purpose of the third key position. If "R", indicates the government region, if "K", indicates the district
- dkm_id: Identifier in the digital landscape model (DLM250)
- ewz: Number of inhabitants
- kf1: Land register area in square kilometers

Boundary geometries ("li") can have additional columns:

- agz: Type of border. Can be one of
 - 1: National border
 - 2: State border
 - 3: Governmental district border
 - 4: District border
 - 5: Administrative association border
 - 6: Municipality border
 - 9: Coastline
- rdg: Legal definition of a border. Can be 1 (determined), 2 (not determined) or 9 (coastline)
- gm5: Border characteristic of administrative association borders (AGZ 5). Used to describe the purpose of these borders. Can be 0 (characteristics by AGZ) or 8 (non-association border)
- gmk: Border characteristic by coast/ocean. Specifies whether a border runs along a waterbody. Can be one of
 - 7: borders on the ocean
 - 8: auxiliary borders on the ocean
 - 9: borders at the coastline
 - 0: no characteristics
- dlm_id: Identifier in the digital landscape model (DLM250)

Point geometries ("pk") have the following additional columns:

- ot1: Name of the locality in the digital landscape model (DLM250)
- lon_dez: Decimal longitude
- lat_dez: Decimal latitude
- lon_gms: Geographical longitude
- lat_gms: Geographical latitude

Query language

By default, WFS requests use CQL (Contextual Query Language) queries for simplicity. CQL queries only work together with GET requests. This means that when the URL is longer than 2048 characters, they fail. While POST requests are much more flexible and able to accommodate long queries, XML is really a pain to work with and I'm not confident in my approach to construct XML queries. You can control whether to send GET or POST requests by setting `options(ffm_query_language = "XML")` or `options(ffm_query_language = "CQL")`.

See Also

[vg250-ew documentation](#)

[vg250-ew MIS record](#)

[bkg_nuts](#) for retrieving EU administrative areas

[bkg_admin_hierarchy](#) for the administrative hierarchy

[bkg_ror](#), [bkg_grid](#), [bkg_kfz](#), [bkg_authorities](#) for non-administrative regions

Datasets: [admin_data](#), [nuts_data](#)

Examples

```
# You can use R-like operators to query the WFS
bkg_admin(ags %LIKE% "05") # districts in NRW
bkg_admin(sn_l == "05") # does the same thing
bkg_admin(gen %LIKE% "Ber%") # districts starting with Ber*

# To query population and area, the key date must be December 31
bkg_admin(ewz > 500000, key_date = "1231") # districts over 500k people
bkg_admin(kfl <= 100, key_date = "1231") # districts with low land register area

# Using `gf == 9`, you can exclude waterbodies like oceans
states <- bkg_admin(scale = "5000", level = "lan", gf == 9)
plot(states$geometry)

# Download historical data
bkg_admin_archive(scale = "5000", level = "sta", year = "2021")

# Download high-resolution data (takes a long time!)
bkg_admin_highres(level = "lan")
```


Description

Retrieve polygon geometries of municipalities in Germany with details on their relationships to administrative areas of higher levels in the territorial hierarchy. The output of this functions contains the identifiers and names of the NUTS1 to NUTS3 areas that each municipality belongs to.

This function interfaces the vz250 product of the BKG.

Usage

```
bkg_admin_hierarchy(  
  key_date = c("0101", "1231"),  
  year = "latest",  
  timeout = 120,  
  update_cache = FALSE  
)
```

Arguments

key_date	For resolution %in% c("250", "5000"), specifies the key date from which to download administrative data. Can be either "0101" (January 1) or "1231" (December 31). The latter is able to georeference statistical data while the first integrates changes made in the new year. If "1231", population data is attached, otherwise not. Note that population data is not available at all scales (usually 250 and 1000). Defaults to "0101".
year	Version year of the dataset. You can use latest to retrieve the latest dataset version available on the BKG's geodata center. Older versions can be browsed using the archive .
timeout	Timeout value for the data download passed to req_timeout . Adjust this if your internet connection is slow or you are downloading larger datasets.
update_cache	By default, downloaded files are cached in the tempdir() directory of R. When downloading the same data again, the data is not downloaded but instead taken from the cache. Sometimes this can be not the desired behavior. If you want to overwrite the cache, pass TRUE. Defaults to FALSE, i.e. always adopt the cache if possible.

Value

An sf tibble with multipolygon geometries similar to the output of [bkg_admin\(level = "gem"\)](#). The tibble additionally contains columns NUTS*_CODE and NUTS*_NAME giving the identifiers and names of the administrative areas the municipalities belong to.

See Also

[vz250 documentation](#)

Examples

```
bkg_admin_hierarchy()
```

bkg_ags

*Official keys***Description**

Retrieve geographical names associated with official municipality keys and regional keys. To retrieve their polygon geometries, see [bkg_admin](#).

These functions interface the wfs_gnde product of the BKG.

Usage

```
bkg_ags(..., filter = NULL, properties = NULL, max = NULL)
```

```
bkg_ars(..., filter = NULL, properties = NULL, max = NULL)
```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

`filter` A character string containing a valid CQL or XML filter. This string is appended to the query constructed through ... Use this argument to construct more complex filters. Defaults to NULL.

`properties` Vector of columns to include in the output.

`max` Maximum number of results to return.

Value

A dataframe containing the respective identifier and geographical names related to their state, government region, district and municipality. `bkg_ars` additionally returns the name of the administrative association.

Query language

While other WFS interfaces like `bkg_admin` allow querying using CQL or XML, `bkg_ags` and `bkg_ars` (using the GNDE service) ONLY support XML. This has implications for the allowed query filters (see `wfs_filter`).

See Also

[wfs_gnde MIS record](#)

[wfs_gnde documentation](#)

[bkg_geonames](#) and [bkg_endonyms](#) for geographical names

Examples

```
# Either get geographical names for identifiers
bkg_ars(ars == "01")

# ... or identifiers for geographical names
bkg_ars(gemeinde == "Köln")
```

bkg_airports	<i>Airports</i>
--------------	-----------------

Description

Retrieve international, regional, and special airports in Germany. Small landing sites are not included.

These functions interface the `wfs_poi_open` product of the BKG.

Usage

```
bkg_airports(
  ...,
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  properties = NULL,
  max = NULL
)
```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

<code>bbox</code>	An sf geometry or a boundary box vector of the format <code>c(xmin, ymin, xmax, ymax)</code> . Used as a geometric filter to include only those geometries that relate to <code>bbox</code> according to the predicate specified in <code>predicate</code> . If an sf geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
<code>poly</code>	An sf geometry. Used as a geometric filter to include only those geometries that relate to <code>poly</code> according to the predicate specified in <code>predicate</code> . Coordinates are automatically transformed to EPSG:25832 (the default CRS).
<code>predicate</code>	A spatial predicate that is used to relate the output geometries with the object specified in <code>bbox</code> or <code>poly</code> . For example, if <code>predicate = "within"</code> , and <code>bbox</code> is specified, returns only those geometries that lie within <code>bbox</code> . Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within", "contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
<code>filter</code>	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through ... Use this argument to construct more complex filters. Defaults to NULL.
<code>epsg</code>	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(...)\$epsg</code> on a spatial object that you are working with. Defaults to 3035.
<code>properties</code>	Vector of columns to include in the output.
<code>max</code>	Maximum number of results to return.

Value

A dataframe containing the following columns:

- name: Geographical name of the POI
- gemeinde: Municipality name
- verwaltung: Administrative association name
- kreis: District name
- regierungs: Government region name
- bundesland: Federal state name
- poi_id: Unique primary key of a point of interest
- icao_code: ICAO code of the airport
- typ: Type of airport. Can be one of the following:
 - international: International airport
 - regional: Regional airport
 - Sonderflughafen: Special airport

Query language

By default, WFS requests use CQL (Contextual Query Language) queries for simplicity. CQL queries only work together with GET requests. This means that when the URL is longer than 2048 characters, they fail. While POST requests are much more flexible and able to accommodate long queries, XML is really a pain to work with and I'm not confident in my approach to construct XML queries. You can control whether to send GET or POST requests by setting `options(ffm_query_language = "XML")` or `options(ffm_query_language = "CQL")`.

See Also

[wfs_poi_open documentation](#)

[wfs_poi_open MIS record](#)

Other points of interest: [bkg_crossings\(\)](#), [bkg_heliports\(\)](#), [bkg_kilometrage\(\)](#), [bkg_seaports\(\)](#), [bkg_stations\(\)](#), [bkg_trauma_centers\(\)](#)

Examples

```
# Get all airports in NRW
airports <- bkg_airports(ars %LIKE% "05%")
nrw <- bkg_admin(level = "lan", sn_l == "05")
plot(nrw$geometry)
plot(airports$geometry, add = TRUE, pch = 16)
```

bkg_area_codes	<i>Area code regions</i>
----------------	--------------------------

Description

Retrieves area code regions (*Vorwahlgebiete*) in Germany. Area code regions are based on the number of registered telephone numbers.

This function interfaces the wfs_gnde product of the BKG.

Usage

```
bkg_area_codes(
  ...,
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  max = NULL
)
```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

`bbox` An sf geometry or a boundary box vector of the format `c(xmin, ymin, xmax, ymax)`. Used as a geometric filter to include only those geometries that relate

	to bbox according to the predicate specified in predicate. If an sf geometry is provided, coordinates are automatically transformed to ESPG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
poly	An sf geometry. Used as a geometric filter to include only those geometries that relate to poly according to the predicate specified in predicate. Coordinates are automatically transformed to ESPG:25832 (the default CRS).
predicate	A spatial predicate that is used to relate the output geometries with the object specified in bbox or poly. For example, if predicate = "within", and bbox is specified, returns only those geometries that lie within bbox. Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within", "contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
filter	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through Use this argument to construct more complex filters. Defaults to NULL.
epsg	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(. . .)\$epsg</code> on a spatial object that you are working with. Defaults to 3035.
max	Maximum number of results to return.

Value

An sf dataframe containing polygon geometries and the area code (vorwahl) associated with the region.

Query language

While other WFS interfaces like [bkg_admin](#) allow querying using CQL or XML, `bkg_area_codes` ONLY supports XML. This has implications for the allowed query filters (see [wfs_filter](#)).

See Also

[wfs_gnde MIS record](#)

[wfs_gnde documentation](#)

[bkg_geonames](#) and [bkg_endonyms](#) for geographical names

Other non-administrative regions: [bkg_authorities\(\)](#), [bkg_grid](#), [bkg_kfz\(\)](#), [bkg_ror\(\)](#)

Examples

```
vorwahlen <- bkg_area_codes(vorwahl %LIKE% "0215%")
plot(vorwahlen$geometry)
```

bkg_authorities *Regions of authority*

Description

Retrieve regions of administrative responsibility for job centers, employment agencies, offices of employment agencies, regional directorates of the Federal Employment Agency as well as local, regional, and higher regional courts.

This function interfaces the wfs_bzb-open product of the BKG.

Usage

```
bkg_authorities(
  authority,
  ...,
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  properties = NULL,
  max = NULL
)
```

Arguments

authority	Type of authority for which to retrieve regions of responsibility. Must be one of "employment_agencies", "employment_offices", "job_centers", "directorates", "local_courts", "regional_courts", or "higher_regional_courts".
...	Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

<code>bbox</code>	An sf geometry or a boundary box vector of the format <code>c(xmin, ymin, xmax, ymax)</code> . Used as a geometric filter to include only those geometries that relate to <code>bbox</code> according to the predicate specified in <code>predicate</code> . If an sf geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
<code>poly</code>	An sf geometry. Used as a geometric filter to include only those geometries that relate to <code>poly</code> according to the predicate specified in <code>predicate</code> . Coordinates are automatically transformed to EPSG:25832 (the default CRS).
<code>predicate</code>	A spatial predicate that is used to relate the output geometries with the object specified in <code>bbox</code> or <code>poly</code> . For example, if <code>predicate = "within"</code> , and <code>bbox</code> is specified, returns only those geometries that lie within <code>bbox</code> . Can be one of <code>"equals"</code> , <code>"disjoint"</code> , <code>"intersects"</code> , <code>"touches"</code> , <code>"crosses"</code> , <code>"within"</code> , <code>"contains"</code> , <code>"overlaps"</code> , <code>"relate"</code> , <code>"dwithin"</code> , or <code>"beyond"</code> . Defaults to <code>"intersects"</code> .
<code>filter</code>	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through <code>...</code> . Use this argument to construct more complex filters. Defaults to <code>NULL</code> .
<code>epsg</code>	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(...)\$epsg</code> on a spatial object that you are working with. Defaults to <code>3035</code> .
<code>properties</code>	Vector of columns to include in the output.
<code>max</code>	Maximum number of results to return.

Value

An sf tibble with multipolygon geometries and the following columns:

- `id`: Identifier of the authority region
- `dst_id`: Identifier of the authority office
- `uebergeord`: Name of the superior authority
- `name`: Name of the authority

Query language

By default, WFS requests use CQL (Contextual Query Language) queries for simplicity. CQL queries only work together with GET requests. This means that when the URL is longer than 2048 characters, they fail. While POST requests are much more flexible and able to accommodate long queries, XML is really a pain to work with and I'm not confident in my approach to construct XML queries. You can control whether to send GET or POST requests by setting `options(ffm_query_language = "XML")` or `options(ffm_query_language = "CQL")`.

See Also

[bzb-open documentation](#)

[bzb-open MIS record](#)

Other non-administrative regions: [bkg_area_codes\(\)](#), [bkg_grid](#), [bkg_kfz\(\)](#), [bkg_ror\(\)](#)

Examples

```
# Get only local courts that are subordinates of the regional court Cottbus
bkg_authorities(
  authority = "local_courts",
  uebergeord %LIKE% "%Cottbus",
  uebergeord %LIKE% "Landgericht%"
)
```

bkg_clc

Corine Land Cover

Description

Retrieve land cover polygons in Germany based on the Corine Land Cover (CLC) nomenclature. [Corine Land Cover](#) is a way to project by the European Commission to consistently classify both land cover and land use.

This function interfaces the wfs_clc5_* products of the BKG.

Usage

```
bkg_clc(
  ...,
  year = "2018",
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  properties = NULL,
  max = NULL
)
```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using options (`ffm_query_language = "xml"`). See also [wfs_filter](#).

<code>year</code>	Version year of the dataset. You can use <code>latest</code> to retrieve the latest dataset version available on the BKG's geodata center. Older versions can be browsed using the archive .
<code>bbox</code>	An sf geometry or a boundary box vector of the format <code>c(xmin, ymin, xmax, ymax)</code> . Used as a geometric filter to include only those geometries that relate to <code>bbox</code> according to the predicate specified in <code>predicate</code> . If an sf geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
<code>poly</code>	An sf geometry. Used as a geometric filter to include only those geometries that relate to <code>poly</code> according to the predicate specified in <code>predicate</code> . Coordinates are automatically transformed to EPSG:25832 (the default CRS).
<code>predicate</code>	A spatial predicate that is used to relate the output geometries with the object specified in <code>bbox</code> or <code>poly</code> . For example, if <code>predicate = "within"</code> , and <code>bbox</code> is specified, returns only those geometries that lie within <code>bbox</code> . Can be one of <code>"equals"</code> , <code>"disjoint"</code> , <code>"intersects"</code> , <code>"touches"</code> , <code>"crosses"</code> , <code>"within"</code> , <code>"contains"</code> , <code>"overlaps"</code> , <code>"relate"</code> , <code>"dwithin"</code> , or <code>"beyond"</code> . Defaults to <code>"intersects"</code> .
<code>filter</code>	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through <code>...</code> . Use this argument to construct more complex filters. Defaults to <code>NULL</code> .
<code>epsg</code>	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(...)\$epsg</code> on a spatial object that you are working with. Defaults to <code>3035</code> .
<code>properties</code>	Vector of columns to include in the output.
<code>max</code>	Maximum number of results to return.

Value

An sf dataframe with polygon geometries and the following columns:

- `clc*`: CLC land cover classes for the given year. An overview of all CLC classes can be found in the [Copernicus documentation](#).
- `shape_length`: Circumference of the polygon in meters
- `shape_area`: Area of the polygon in square meters

Query language

By default, WFS requests use CQL (Contextual Query Language) queries for simplicity. CQL queries only work together with GET requests. This means that when the URL is longer than 2048 characters, they fail. While POST requests are much more flexible and able to accommodate long queries, XML is really a pain to work with and I'm not confident in my approach to construct XML queries. You can control whether to send GET or POST requests by setting `options(ffm_query_language = "XML")` or `options(ffm_query_language = "CQL")`.

See Also

[wfs_clc5_2018 documentation](#)

[wfs_clc5_2018 MIS record](#)

Examples

```
# Get glaciers in Germany
bkg_clc(clc18 == "335")

# Get all coastal wetlands
bkg_clc(clc18 %LIKE% "42%")

# Get only wetlands in Lower Saxony
lowsax <- bkg_admin(level = "lan", scale = "5000", sn_l == "03", gf == 9)
wetlands <- bkg_clc(clc18 %LIKE% "4%", poly = lowsax, lang = "xml")
plot(lowsax$geometry)
plot(wetlands$geometry, add = TRUE)
```

bkg_crossings

Border crossings

Description

Retrieve border crossings in Germany. A road is a border crossing if it touches an international border and it continues on the foreign side. This includes ferry connections but not dirt roads.

Usage

```
bkg_crossings(
  ...,
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  properties = NULL,
  max = NULL
)
```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

<code>bbox</code>	An sf geometry or a boundary box vector of the format <code>c(xmin, ymin, xmax, ymax)</code> . Used as a geometric filter to include only those geometries that relate to <code>bbox</code> according to the predicate specified in <code>predicate</code> . If an sf geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
<code>poly</code>	An sf geometry. Used as a geometric filter to include only those geometries that relate to <code>poly</code> according to the predicate specified in <code>predicate</code> . Coordinates are automatically transformed to EPSG:25832 (the default CRS).
<code>predicate</code>	A spatial predicate that is used to relate the output geometries with the object specified in <code>bbox</code> or <code>poly</code> . For example, if <code>predicate = "within"</code> , and <code>bbox</code> is specified, returns only those geometries that lie within <code>bbox</code> . Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within", "contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
<code>filter</code>	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through ... Use this argument to construct more complex filters. Defaults to NULL.
<code>epsg</code>	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(...)\$epsg</code> on a spatial object that you are working with. Defaults to 3035.
<code>properties</code>	Vector of columns to include in the output.
<code>max</code>	Maximum number of results to return.

Value

A dataframe with the following columns:

- name: Geographical name of the POI
- gemeinde: Municipality name
- verwaltung: Administrative association name
- kreis: District name
- regierungs: Government region name
- bundesland: Federal state name
- ort: Name of the nearest place
- strasse: Number or label of the border-crossing street
- typ: Type of checkpoint; always "Straßenverkehr"

Query language

By default, WFS requests use CQL (Contextual Query Language) queries for simplicity. CQL queries only work together with GET requests. This means that when the URL is longer than 2048 characters, they fail. While POST requests are much more flexible and able to accommodate long queries, XML is really a pain to work with and I'm not confident in my approach to construct XML queries. You can control whether to send GET or POST requests by setting `options(ffm_query_language = "XML")` or `options(ffm_query_language = "CQL")`.

See Also

[wfs_poi_open documentation](#)

[wfs_poi_open MIS record](#)

Other points of interest: [bkg_airports\(\)](#), [bkg_heliports\(\)](#), [bkg_kilometrage\(\)](#), [bkg_seaports\(\)](#), [bkg_stations\(\)](#), [bkg_trauma_centers\(\)](#)

Examples

```
# Get all border crossings in Bavaria
crossings <- bkg_crossings(bundesland == "Bayern")
plot(crossings$geometry, pch = 16)
```

bkg_dem

Digital elevation model

Description

Retrieve the digital elevation model (DEM) for the territory of Germany.

This function interfaces the `wcs_dgm200_inspire` product of the BKG.

Usage

```
bkg_dem(bbox = NULL, interpolation = NULL, epsg = 3035)
```

Arguments

bbox An sf geometry or a boundary box vector of the format `c(xmin, ymin, xmax, ymax)`. Used as a geometric filter to mask the coverage raster. If an sf geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.

interpolation Interpolation method to preprocess the raster. Can be "nearest-neighbor", "linear", or "cubic". Does not seem to work currently - despite being listed as a capability of the WCS.

epsg An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running `sf::st_crs(...)$epsg` on a spatial object that you are working with. Defaults to 3035.

Value

A [SpatRaster](#) containing elevation data.

See Also

[wcs_dgm200_inspire documentation](#)

[wcs_dgm200_inspire MIS record](#)

Examples

```
library(sf)

# Elevation around Hanover
han <- st_sfc(st_point(c(9.738611, 52.374444)), crs = 4326)
han <- st_buffer(st_transform(han, 3035), dist = 2000)
dem <- bkg_dem(bbox = han)
terra::plot(dem)
```

bkg_dlm

Digital landscape model (DLM)

Description

Retrieve objects from the digital landscape model (DLM). DLMs are a description of topographical objects of a landscape. Many other services from the BKG are derived from the DLM.

Although this function lets you download each feature type in the DLM, you still need to know about what data is available and what the features in the output actually mean. Since the DLM gets pretty complicated, you are advised to take a look at the [GeoInfoDok](#) object type catalog.

This function interfaces the `d1m*` products of the BKG.

Usage

```

bkg_dlm(
  type,
  ...,
  shape = c("point", "line", "polygon"),
  scale = c("250", "1000"),
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  properties = NULL,
  max = NULL
)

```

Arguments

type Feature type of the DLM. Can either be the identifier (e.g., 41010) or its description (e.g., Siedlungsflaeche). The description can either be prefixed with AX_ or not. Providing an identifier directly is generally faster as the description needs to be matched by requesting the `GetCapabilities` endpoint of the service.

Note that not all feature types are available for all shapes (see the `shape` argument). To see all available feature types, you can run `bkg_feature_types("dlm250")` or `bkg_feature_types("dlm1000")`.

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

shape Geometry type of the feature type. Must be one of "point", "line", or "polygon". Defaults to "point". Not all shapes are available for all feature types.

scale	Scale of the geometries. Can be "250" (1:250,000) or "1000" (1:1,000,000). Defaults to "250".
bbox	An sf geometry or a boundary box vector of the format c(xmin, ymin, xmax, ymax). Used as a geometric filter to include only those geometries that relate to bbox according to the predicate specified in predicate. If an sf geometry is provided, coordinates are automatically transformed to ESPG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
poly	An sf geometry. Used as a geometric filter to include only those geometries that relate to poly according to the predicate specified in predicate. Coordinates are automatically transformed to ESPG:25832 (the default CRS).
predicate	A spatial predicate that is used to relate the output geometries with the object specified in bbox or poly. For example, if predicate = "within", and bbox is specified, returns only those geometries that lie within bbox. Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within", "contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
filter	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through Use this argument to construct more complex filters. Defaults to NULL.
epsg	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running sf::st_crs(. . .)\$epsg on a spatial object that you are working with. Defaults to 3035.
properties	Vector of columns to include in the output.
max	Maximum number of results to return.

Value

An sf tibble with the geometry suggested by shape. The columns can vary depending of the selected feature type. The meanings of the columns can also change depending on the feature type. Check out the GeoInfoDok object type catalog for a detailed documentation of the DLM metadata. Some more general columns are included for all feature types; these include:

- id: Identifier of an object
- land: ISO-2 code of the country, usually DE
- modellart: Model type
- objart: Feature type of the digital landscape model (DLM)
- objart_txt: Title of the feature type
- objid: Unique object identifier
- beginn: Creation of the object in the DLM
- ende: Deletion of the object from the DLM
- objart_z: Object type of the composite object (ZUSO)
- objid_z: Object type of the composite object (ZUSO)

Query language

By default, WFS requests use CQL (Contextual Query Language) queries for simplicity. CQL queries only work together with GET requests. This means that when the URL is longer than 2048 characters, they fail. While POST requests are much more flexible and able to accommodate long queries, XML is really a pain to work with and I'm not confident in my approach to construct XML queries. You can control whether to send GET or POST requests by setting `options(ffm_query_language = "XML")` or `options(ffm_query_language = "CQL")`.

See Also

[d1m250 documentation](#)

[d1m250 MIS record](#)

Examples

```
# Retrieve all train tracks in Leipzig
library(sf)
lzg <- st_sfc(st_point(c(12.37475, 51.340333)), crs = 4326)
lzg <- st_buffer(st_transform(lzg, 3035), dist = 10000, endCapStyle = "SQUARE")

tracks <- bkg_d1m("Bahnstrecke", shape = "line", poly = lzg)
tracks

plot(lzg)
plot(tracks$geometry, add = TRUE)

# Filter all tracks that are not rail cargo
bkg_d1m("Bahnstrecke", shape = "line", poly = lzg, bkt == "1102")

# Directly providing the identifier is faster
bkg_d1m("42014", shape = "line", poly = lzg)
```

bkg_geonames

Geographical objects and endonyms

Description

Get geographic names including toponyms and endonyms. `bkg_geonames` retrieves the geographical "objects" based on the digital landscape model (DLM). These objects contain a set of metadata and a national name identifier (NNID). These NNIDs can be used to join with the endonyms related to a geographical object (`bkg_endonyms`).

These functions interface the `wfs_gnde` product of the BKG.

Usage

```

bkg_geonames(
  ...,
  names = TRUE,
  ags = FALSE,
  dlm = FALSE,
  status = FALSE,
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  properties = NULL,
  max = NULL
)

bkg_endonyms(..., filter = NULL, properties = NULL, max = NULL)

```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

names If TRUE, includes endonyms of the geographical objects in the output using `bkg_endonyms`. Technically, this can be FALSE, because the endpoint only returns meta data on geographical names by default. If this argument is TRUE, the output is merged with the endonym table requiring an additional request. Defaults to TRUE.

ags If TRUE, resolves AGS codes to geographical names using [bkg_ags](#). Note that setting this to TRUE requires an additional web request. Defaults to FALSE.

d1m	If TRUE, adds the DLM identifier corresponding to the national name identifiers (NNID) of the output using <code>bkg_d1m</code> . Note that setting this to TRUE requires an additional web request. Defaults to FALSE.
status	If TRUE, adds the date of the objects last edit to the output. Note that setting this to TRUE requires an additional web request. Defaults to FALSE.
bbox	An sf geometry or a boundary box vector of the format <code>c(xmin, ymin, xmax, ymax)</code> . Used as a geometric filter to include only those geometries that relate to <code>bbox</code> according to the predicate specified in <code>predicate</code> . If an sf geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
poly	An sf geometry. Used as a geometric filter to include only those geometries that relate to <code>poly</code> according to the predicate specified in <code>predicate</code> . Coordinates are automatically transformed to EPSG:25832 (the default CRS).
predicate	A spatial predicate that is used to relate the output geometries with the object specified in <code>bbox</code> or <code>poly</code> . For example, if <code>predicate = "within"</code> , and <code>bbox</code> is specified, returns only those geometries that lie within <code>bbox</code> . Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within", "contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
filter	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through <code>...</code> . Use this argument to construct more complex filters. Defaults to NULL.
epsg	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(...)\$epsg</code> on a spatial object that you are working with. Defaults to 3035.
properties	Vector of columns to include in the output.
max	Maximum number of results to return.

Details

These functions make use of the GN-DE WFS, just like `bkg_ags`, `bkg_ars`, and `bkg_area_codes`. The infrastructure behind it is actually quite sophisticated and this function may not live up to these standards. You can use `bkg_feature_types` and `bkg_wfs` to manually explore the service's endpoints if required.

Value

A dataframe containing the following columns:

- `nnid`: National name identifier
- `landesCode`: Country identifier; 276 is Germany.
- `beschreibung`: Optional details
- `geoLaenge`: Geographical longitude
- `geoBreite`: Geographical latitude
- `hoehe`: Elevation above sea level

- `hoeheger`: Computed elevation above sea level
- `groesse`: Undocumented, but I guess this relates to the suggested print size of the labels
- `ewz`: Number of inhabitants
- `ewzger`: Computed number of inhabitants
- `ags`: Official municipality key (Amtlicher Gemeindeschlüssel). Related to the ARS but shortened to omit position 6 to 9. Structured as follows:
 - Position 1-2: Federal state
 - Position 3: Government region
 - Position 4-5: District
 - Position 6-8: Municipality
- `gemteil`: Whether the place is part of a municipality
- `virtuell`: Whether the place is a real or virtual locality
- `ars`: Territorial code (Amtlicher Regionalschlüssel). The ARS is structured hierarchically as follows:
 - Position 1-2: Federal state
 - Position 3: Government region
 - Position 4-5: District
 - Position 6-9: Administrative association
 - Position 10-12: Municipality

If `ags = TRUE`, adds the output of [bkg_ags](#). If `d1m = TRUE`, adds a column `d1m_id` containing identifiers of [bkg_d1m](#).

`bkg_endonyms` contains the following columns:

- `name`: Name of the geographical object
- `geschlecht`: If applicable, the grammatical gender of a geographical name

These are also included in the output of `bkg_geonames` if `names = TRUE`.

Query language

While other WFS interfaces like [bkg_admin](#) allow querying using CQL or XML, `bkg_geonames` and `bkg_endonyms` (using the GNDE service) ONLY support XML. This has implications for the allowed query filters (see [wfs_filter](#)).

See Also

[wfs_gnde MIS record](#)

[wfs_gnde documentation](#)

[bkg_ags](#) and [bkg_ars](#) for geographical names of administrative areas

Examples

```
# Plot geographical objects in Cologne
library(sf)
library(ggplot2)
cgn <- st_sfc(st_point(c(6.956944, 50.938056)), crs = 4326)
cgn <- st_buffer(st_transform(cgn, 3035), dist = 500)

cgn_names <- bkg_geonames(poly = cgn)
st_geometry(cgn_names) <- st_centroid(st_geometry(cgn_names))
cgn_names <- cgn_names[lengths(st_intersects(cgn_names, cgn)) > 0, ]
ggplot(cgn_names) + geom_sf_text(aes(label = name)) + theme_void()
```

bkg_grid

INSPIRE grids

Description

Retrieve geometries of INSPIRE-compliant grid geometries (also called "GeoGitter"). `bkg_grid_fast()` is much faster than `bkg_grid_full()` by downloading heavily compressed versions grids. This happens at the cost of data richness as `bkg_grid_fast()` only contains the geometries and nothing else. Note that the arrow package needs to be installed to use `bkg_grid_fast()`.

Note that the output contains point geometries. Most of the times, you want to work with rasters instead. To convert a given object out, type the following (terra package required):

```
terra::rast(out)
```

These functions interface the GeoGitter product of the BKG.

Usage

```
bkg_grid_fast(
  year = c("2019", "2018", "2017", "2015"),
  resolution = c("100km", "10km", "5km", "1km", "250m", "100m"),
  timeout = 600,
  update_cache = FALSE
)

bkg_grid_full(
  year = "latest",
  resolution = c("100km", "10km", "5km", "1km", "250m", "100m"),
  timeout = 600,
  update_cache = FALSE
)
```

Arguments

year	Version of the grid. Can be "2015", "2017", "2018" or "2019". For <code>bkg_grid_fast</code> , "latest" downloads the latest version of the grid.
resolution	Cell size of the grid. Can be "100m", "250m", "1km", "5km", "10km", or "100km".
timeout	Timeout value for the data download passed to <code>req_timeout</code> . Adjust this if your internet connection is slow or you are downloading larger datasets.
update_cache	By default, downloaded files are cached in the <code>tempdir()</code> directory of R. When downloading the same data again, the data is not downloaded but instead taken from the cache. Sometimes this can be not the desired behavior. If you want to overwrite the cache, pass <code>TRUE</code> . Defaults to <code>FALSE</code> , i.e. always adopt the cache if possible.

Details

The following table gives a rough idea of how much less data `bkg_grid_fast` needs to download for each resolution compared to `bkg_grid_full`.

	Size	fast	full
100km		0.78 kB	933 kB
10km		2.68 kB	1,015 kB
5km		3.53 kB	1,253 kB
1km		28.7 kB	5,249 kB
500m		133 kB	15,902 kB
250m		289 kB	53,900 kB
100m		1,420 kB	291,000 kB

Value

`bkg_grid_fast` returns an sf dataframe with point geometries and no features. `bkg_grid_full` also returns point geometries but with the following additional features:

- `x_sw`: X coordinate of the South-West corner of a grid cell
- `y_sw`: Y coordinate of the South-West corner of a grid cell
- `f_staat`: State area in the grid cell in square meters
- `f_land`: Land area in the grid cell in square meters
- `f_wasser`: Water area in the grid cell in square meters
- `p_staat`: Share of state area in the grid cell
- `p_land`: Share of land area in the grid cell
- `p_wasser`: Share of water area in the grid cell
- `ags`: Official municipality key (Amtlicher Gemeindeschlüssel). Related to the ARS but shortened to omit position 6 to 9. Structured as follows:
 - Position 1-2: Federal state
 - Position 3: Government region
 - Position 4-5: District
 - Position 6-8: Municipality

Note that `ags` is only included for resolutions "100m" and "250m"

See Also

[GeoGitter documentation](#)

[GeoGitter MIS record](#)

Other non-administrative regions: [bkg_area_codes\(\)](#), [bkg_authorities\(\)](#), [bkg_kfz\(\)](#), [bkg_ror\(\)](#)

Examples

```
# Return a bare-bones version of the INSPIRE grid
grid <- bkg_grid_fast(year = "2019", resolution = "100km")

# Return a fully detailed version instead
grid_full <- bkg_grid_full(resolution = "5km")

plot(grid)

# Convert grid to SpatRaster
if (requireNamespace("terra")) {
  library(terra)
  raster <- rast(vect(grid_full["p_wasser"]), type = "xyz")
  plot(raster, main = "Share of water area")
}
```

bkg_heliports

Heliports

Description

Get heliports in Germany. Based on data from third-party providers and image classification of aerial imagery.

Usage

```
bkg_heliports(
  ...,
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  properties = NULL,
  max = NULL
)
```


Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

<code>bbox</code>	An sf geometry or a boundary box vector of the format <code>c(xmin, ymin, xmax, ymax)</code> . Used as a geometric filter to include only those geometries that relate to <code>bbox</code> according to the predicate specified in <code>predicate</code> . If an sf geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
<code>poly</code>	An sf geometry. Used as a geometric filter to include only those geometries that relate to <code>poly</code> according to the predicate specified in <code>predicate</code> . Coordinates are automatically transformed to EPSG:25832 (the default CRS).
<code>predicate</code>	A spatial predicate that is used to relate the output geometries with the object specified in <code>bbox</code> or <code>poly</code> . For example, if <code>predicate = "within"</code> , and <code>bbox</code> is specified, returns only those geometries that lie within <code>bbox</code> . Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within", "contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
<code>filter</code>	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through ... Use this argument to construct more complex filters. Defaults to NULL.
<code>epsg</code>	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(...)\$epsg</code> on a spatial object that you are working with. Defaults to 3035.
<code>properties</code>	Vector of columns to include in the output.
<code>max</code>	Maximum number of results to return.

Value

A dataframe with the following columns:

- name: Geographical name of the POI
- gemeinde: Municipality name
- verwaltung: Administrative association name
- kreis: District name
- regierungs: Government region name
- bundesland: Federal state name
- code: Identifier of the heliport
- name_bkg: Name of the landing site according to BKG
- name_dfs: Name of the landing size according to Deutsche Flugsicherung (DFS)
- airport_pk: Identifier according to the LFS aviation manual
- befestigung: Pavement type of the landing site. Can be:
 - befestigt: paved
 - teilweise befestigt: partially paved
 - unbefestigt: unpaved
- kennzeichen: Marking of the landing size. Can be:
 - gekennzeichnet: marked
 - nicht gekennzeichnet: not marked
- lage: Location of the landing size. Can be:
 - D: Roof
 - F: Field
 - PG: Platform next to a hospital
 - W: Pasture
 - LP: Landing site
 - PP: Parking lot
 - LP / W: Paved landing size on pasture
 - F / W: Field or pasture
 - LP / Str.: Landing size next to a street
- typ: Type of heliport. Can be:
 - H: Heliport
 - HH: Heliport at a hospital
 - MH: Military heliport
- typ2: Additional heliport type for landing sites with an air rescue station. Can be:
 - HRLS: Helicopter air rescue station
 - ITH: Intensive transport helicopter
- betreiber: Operator of the heliport
- helikopter: Name of the helicopter belonging to the air rescue station

- status: Whether the point geometry was edited by the BKG. Can be:
 - Original: not edited
 - Vershoben: moved
 - neu: newly added
- quelle: Source of the information. Can be:
 - BKG: Own research by the BKG
 - DFS-Liste: Provided by DFS
 - LBA-Liste: Provided by the Federal Aviation Office (LBA)
 - MHW: Provided by the Medical Disaster Relief Organization (MHW)
 - RTH.Info: Provided by rth.info

Query language

By default, WFS requests use CQL (Contextual Query Language) queries for simplicity. CQL queries only work together with GET requests. This means that when the URL is longer than 2048 characters, they fail. While POST requests are much more flexible and able to accommodate long queries, XML is really a pain to work with and I'm not confident in my approach to construct XML queries. You can control whether to send GET or POST requests by setting `options(ffm_query_language = "XML")` or `options(ffm_query_language = "CQL")`.

See Also

[wfs_poi_open documentation](#)

[wfs_poi_open MIS record](#)

Other points of interest: [bkg_airports\(\)](#), [bkg_crossings\(\)](#), [bkg_kilometrage\(\)](#), [bkg_seaports\(\)](#), [bkg_stations\(\)](#), [bkg_trauma_centers\(\)](#)

Examples

```
# Get only military heliports
bkg_heliports(typ == "MH")

# Get only rooftop heliports
bkg_heliports(lage == "D")
```

bkg_kfz

Vehicle registration plates

Description

Retrieve motor vehicle registration plate regions in Germany. Registration plate regions are discerned by their area code (*Unterscheidungszeichen*) which indicate the place where a vehicle was registered. These regions partially overlap with districts but are not entirely identical.

This function interfaces the `wfs_kfz250` product of the BKG.

Usage

```

bkg_kfz(
  ...,
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  properties = NULL,
  max = NULL
)

```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

bbox An sf geometry or a boundary box vector of the format `c(xmin, ymin, xmax, ymax)`. Used as a geometric filter to include only those geometries that relate to `bbox` according to the predicate specified in `predicate`. If an sf geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.

poly An sf geometry. Used as a geometric filter to include only those geometries that relate to `poly` according to the predicate specified in `predicate`. Coordinates are automatically transformed to EPSG:25832 (the default CRS).

predicate A spatial predicate that is used to relate the output geometries with the object specified in `bbox` or `poly`. For example, if `predicate = "within"`, and `bbox` is specified, returns only those geometries that lie within `bbox`. Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within",

	"contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
filter	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through Use this argument to construct more complex filters. Defaults to NULL.
epsg	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(. . .)\$epsg</code> on a spatial object that you are working with. Defaults to 3035.
properties	Vector of columns to include in the output.
max	Maximum number of results to return.

Value

An sf dataframe with multipolygon geometries and the following columns:

- debkgid: Identifier in the digital landscape model DLM250
- nnid: National name identifier
- name: Name of the geographical object
- ars: Territorial code (Amtlicher Regionalschlüssel). The ARS is structured hierarchically as follows:
 - Position 1-2: Federal state
 - Position 3: Government region
 - Position 4-5: District
 - Position 6-9: Administrative association
 - Position 10-12: Municipality
- oba: Name of the ATKIS object type
- kfz: Vehicle registration area code, comma-separated in case of multiple codes
- geola: Geographical longitude
- geobr: Geographical latitude
- gkre: Gauß-Krüger easting
- gkho: Gauß-Krüger northing
- utmre: UTM easting
- utmho: UTM northing

See Also

[kfz250 documentation](#)

[kfz250 MIS record](#)

[bkg_admin](#)

Other non-administrative regions: [bkg_area_codes\(\)](#), [bkg_authorities\(\)](#), [bkg_grid](#), [bkg_ror\(\)](#)

Examples

```
library(ggplot2)

kfz <- bkg_kfz(ars %LIKE% "053%")
ggplot(kfz) +
  geom_sf(fill = NA) +
  geom_sf_text(aes(label = kfz)) +
  theme_void()
```

bkg_kilometrage

Kilometrage

Description

Get kilometrages of German federal motorways. Kilometrages are markers for each kilometer of a highway. They can be used to create linear referencing systems (LRS).

Usage

```
bkg_kilometrage(
  ...,
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  properties = NULL,
  max = NULL
)
```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo

%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

<code>bbox</code>	An sf geometry or a boundary box vector of the format <code>c(xmin, ymin, xmax, ymax)</code> . Used as a geometric filter to include only those geometries that relate to <code>bbox</code> according to the predicate specified in <code>predicate</code> . If an sf geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
<code>poly</code>	An sf geometry. Used as a geometric filter to include only those geometries that relate to <code>poly</code> according to the predicate specified in <code>predicate</code> . Coordinates are automatically transformed to EPSG:25832 (the default CRS).
<code>predicate</code>	A spatial predicate that is used to relate the output geometries with the object specified in <code>bbox</code> or <code>poly</code> . For example, if <code>predicate = "within"</code> , and <code>bbox</code> is specified, returns only those geometries that lie within <code>bbox</code> . Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within", "contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
<code>filter</code>	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through <code>...</code> . Use this argument to construct more complex filters. Defaults to NULL.
<code>epsg</code>	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(...)\$epsg</code> on a spatial object that you are working with. Defaults to 3035.
<code>properties</code>	Vector of columns to include in the output.
<code>max</code>	Maximum number of results to return.

Value

A dataframe containing the following columns:

- `name`: Geographical name of the POI
- `gemeinde`: Municipality name
- `verwaltung`: Administrative association name
- `kreis`: District name
- `regierungs`: Government region name
- `bundesland`: Federal state name
- `bez`: Label of the federal motorway
- `kilometer`: Kilometrage of the motorway
- `richtung`: Direction of the kilometrage

Query language

By default, WFS requests use CQL (Contextual Query Language) queries for simplicity. CQL queries only work together with GET requests. This means that when the URL is longer than 2048 characters, they fail. While POST requests are much more flexible and able to accommodate long queries, XML is really a pain to work with and I'm not confident in my approach to construct XML queries. You can control whether to send GET or POST requests by setting `options(ffm_query_language = "XML")` or `options(ffm_query_language = "CQL")`.

See Also

[wfs_poi_open documentation](#)

[wfs_poi_open MIS record](#)

The `rLFT` package for linear referencing

Other points of interest: [bkg_airports\(\)](#), [bkg_crossings\(\)](#), [bkg_heliports\(\)](#), [bkg_seaports\(\)](#), [bkg_stations\(\)](#), [bkg_trauma_centers\(\)](#)

Examples

```
# Get the kilometrage of the A2 motorway
a2 <- bkg_kilometrage(bez == "A2")
plot(a2["kilometer"], pch = 16)
```

bkg_nuts

NUTS regions

Description

Retrieve polygons of NUTS regions.

This function interfaces the `nuts*` products of the BKG.

Usage

```
bkg_nuts(
  level = c("1", "2", "3"),
  scale = c("250", "1000", "2500", "5000"),
  key_date = c("0101", "1231"),
  year = "latest",
  timeout = 120,
  update_cache = FALSE
)
```


Arguments

level	NUTS level to download. Can be "1" (federal states), "2" (inconsistent, something between states and government regions), or "3" (districts). Defaults to federal states.
scale	Scale of the geometries. Can be "250" (1:250,000), "1000" (1:1,000,000), "2500" (1:2,500,000) or "5000" (1:5,000,000). If "250", population data is included in the output. Defaults to "250".
key_date	For resolution <code>%in% c("250", "5000")</code> , specifies the key date from which to download administrative data. Can be either "0101" (January 1) or "1231" (December 31). The latter is able to georeference statistical data while the first integrates changes made in the new year. If "1231", population data is attached, otherwise not. Note that population data is not available at all scales (usually 250 and 1000). Defaults to "0101".
year	Version year of the dataset. You can use <code>latest</code> to retrieve the latest dataset version available on the BKG's geodata center. Older versions can be browsed using the archive .
timeout	Timeout value for the data download passed to <code>req_timeout</code> . Adjust this if your internet connection is slow or you are downloading larger datasets.
update_cache	By default, downloaded files are cached in the <code>tempdir()</code> directory of R. When downloading the same data again, the data is not downloaded but instead taken from the cache. Sometimes this can be not the desired behavior. If you want to overwrite the cache, pass <code>TRUE</code> . Defaults to <code>FALSE</code> , i.e. always adopt the cache if possible.

Value

An sf dataframe with multipolygon geometries and the following columns:

- GF: Integer representing the geofactor; whether an area is "structured" or not. Land is structured if it is part of a state or other administrative unit but is not further divided into administrative units. Can be one of
 - 1: Unstructured, waterbody
 - 2: Structured, waterbody
 - 3: Unstructured, land
 - 4: Structured, land
- NUTS_LEVEL: NUTS level. Can be one of
 - 1: NUTS-1; federal states
 - 2: NUTS-2; inconsistent, somewhere between government regions and federal states
 - 3: NUTS-3; districts
- NUTS_CODE: Hierarchical key of the NUTS region. Can have a different number of characters depending on the NUTS level:
 - NUTS-1: three digits
 - NUTS-2: four digits
 - NUTS-3: five digits
- NUTS_NAME: Geographical name of the NUTS region

Note

This function does not query a WFS so you are only able to download entire datasets without the ability to filter beforehand.

See Also

[nuts250 documentation](#)

[nuts250 MIS record](#)

[bkg_admin](#) for retrieving German administrative areas

Datasets: [admin_data](#), [nuts_data](#)

Examples

```
# Download NUTS state data from 2020
bkg_nuts(scale = "5000", year = 2020)

# Download the latest NUTS district data
bkg_nuts(level = "3")
```

bkg_quasigeoid

Quasigeoid

Description

Retrieves the "German Combined Quasigeoid", the official height reference surface of the German land survey above the reference ellipsoid (GRS80).

A quasigeoid is an approximation of the geoid surface used to define normal heights above the earth's surface that is based on more practical assumptions than a true geoid. It defines heights in meters that can be more meaningful than ellipsoidal heights in many applications like surveying, hydrological modeling, engineering, or spatial analysis.

This function interfaces the quasigeoid product of the BKG.

Usage

```
bkg_quasigeoid(
  year = "latest",
  region = c("all", "coast", "no", "nw", "s", "w"),
  timeout = 120,
  update_cache = FALSE
)
```

Arguments

year	Version year of the dataset. You can use <code>latest</code> to retrieve the latest dataset version available on the BKG's geodata center. Older versions can be browsed using the archive .
region	Subterritory of Germany. "all" returns the data for all of Germany, "coast" returns only coastal regions and "no", "nw", "s" and "w" refer to cardinal directions. Defaults to "all".
timeout	Timeout value for the data download passed to <code>req_timeout</code> . Adjust this if your internet connection is slow or you are downloading larger datasets.
update_cache	By default, downloaded files are cached in the <code>tempdir()</code> directory of R. When downloading the same data again, the data is not downloaded but instead taken from the cache. Sometimes this can be not the desired behavior. If you want to overwrite the cache, pass <code>TRUE</code> . Defaults to <code>FALSE</code> , i.e. always adopt the cache if possible.

Value

A `SpatRaster` containing normal heights for the specified region. The data comes in EPSG:4258 and a resolution of 30" x 45" (approximately 0.9 x 0.9 km).

See Also

[quasigeoid documentation](#)

[quasigeoid MIS record](#)

Examples

```
library(terra)
qgeoid <- bkg_quasigeoid(region = "no")
terra::plot(qgeoid)
```

bkg_ror

Non-administrative regions

Description

Retrieve areal data related to what the BKG calls non-administrative regions. This includes:

- `bkg_ror`: Raumordnungsregionen (Spatial planning regions)
- `bkg_rg`: Reisegebiete (Travel areas)
- `bkg_amr`: Arbeitsmarktregionen (Labor market regions)
- `bkg_bkr`: Braunkohlereviere (Lignite regions)
- `bkg_krg`: Kreisregionen (District regions)
- `bkg_mbe`: BBSR Mittelbereiche (BBSR middle areas)

- bkg_ggr: Großstadtregionen (City regions)
- bkg_kmr: Metropolregionen (Metropolitan regions)
- bkg_mkro: Verdichtungsräume (Conurbations)

These functions interface the `ge*` product of the BKG.

Usage

```
bkg_ror(  
  scale = c("250", "1000", "2500", "5000"),  
  year = "latest",  
  timeout = 120,  
  update_cache = FALSE  
)
```

```
bkg_rg(  
  scale = c("250", "1000", "2500", "5000"),  
  year = "latest",  
  timeout = 120,  
  update_cache = FALSE  
)
```

```
bkg_amr(  
  scale = c("250", "1000", "2500", "5000"),  
  year = "latest",  
  timeout = 120,  
  update_cache = FALSE  
)
```

```
bkg_bkr(  
  scale = c("250", "1000", "2500", "5000"),  
  year = "latest",  
  timeout = 120,  
  update_cache = FALSE  
)
```

```
bkg_krg(  
  scale = c("250", "1000", "2500", "5000"),  
  year = "latest",  
  timeout = 120,  
  update_cache = FALSE  
)
```

```
bkg_mbe(  
  scale = c("250", "1000", "2500", "5000"),  
  year = "latest",  
  timeout = 120,  
  update_cache = FALSE  
)
```

```

)

bkg_ggr(
  scale = c("250", "1000", "2500", "5000"),
  year = "latest",
  timeout = 120,
  update_cache = FALSE
)

bkg_kmr(
  scale = c("250", "1000", "2500", "5000"),
  year = "latest",
  timeout = 120,
  update_cache = FALSE
)

bkg_mkro(
  scale = c("250", "1000", "2500", "5000"),
  year = "latest",
  timeout = 120,
  update_cache = FALSE
)

```

Arguments

scale	Scale of the geometries. Can be "250" (1:250,000), "1000" (1:1,000,000), "2500" (1:2,500,000) or "5000" (1:5,000,000). If "250", population data is included in the output. Defaults to "250".
year	Version year of the dataset. You can use latest to retrieve the latest dataset version available on the BKG's geodata center. Older versions can be browsed using the archive .
timeout	Timeout value for the data download passed to <code>req_timeout</code> . Adjust this if your internet connection is slow or you are downloading larger datasets.
update_cache	By default, downloaded files are cached in the <code>tempdir()</code> directory of R. When downloading the same data again, the data is not downloaded but instead taken from the cache. Sometimes this can be not the desired behavior. If you want to overwrite the cache, pass TRUE. Defaults to FALSE, i.e. always adopt the cache if possible.

Value

An sf tibble with multipolygon geometries and two features, a regional identifier and the region endonyms.

See Also

[ge5000 documentation](#)
[ge5000 MIS record](#)

Other non-administrative regions: [bkg_area_codes\(\)](#), [bkg_authorities\(\)](#), [bkg_grid](#), [bkg_kfz\(\)](#)

bkg_seaports	<i>Seaports</i>
--------------	-----------------

Description

Retrieve seaports to the North and Baltic Sea in Northern Germany.

Usage

```
bkg_seaports(
  ...,
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  properties = NULL,
  max = NULL
)
```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

bbox	An sf geometry or a boundary box vector of the format c(xmin, ymin, xmax, ymax). Used as a geometric filter to include only those geometries that relate to bbox according to the predicate specified in predicate. If an sf geometry is provided, coordinates are automatically transformed to ESPG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
poly	An sf geometry. Used as a geometric filter to include only those geometries that relate to poly according to the predicate specified in predicate. Coordinates are automatically transformed to ESPG:25832 (the default CRS).
predicate	A spatial predicate that is used to relate the output geometries with the object specified in bbox or poly. For example, if predicate = "within", and bbox is specified, returns only those geometries that lie within bbox. Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within", "contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
filter	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through Use this argument to construct more complex filters. Defaults to NULL.
epsg	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running sf::st_crs(. . .)\$epsg on a spatial object that you are working with. Defaults to 3035.
properties	Vector of columns to include in the output.
max	Maximum number of results to return.

Value

A dataframe containing the following columns:

- name: Geographical name of the POI
- gemeinde: Municipality name
- verwaltung: Administrative association name
- kreis: District name
- regierungs: Government region name
- bundesland: Federal state name
- poi_id: Unique primary key of a point of interest
- betreiber: Operator of the seaport
- homepage: Homepage of the operator
- typ: Type of seaport. Can be "Seehafen" (seaport) or "See- und Binnenhafen" (sea and inland port)
- art: Type of seaport by freight. Can be:
 - Güter: Goods
 - Güter und Passagiere: Goods and passengers
 - Passagiere: Passengers
- quelle: Source of the information. Can be:
 - BSH: Federal Maritime and Hydrographic Agency
 - MarWiLo: Maritime Wirtschaft & Logistik
 - ZDS-Seehäfen: Zentralverband der deutschen Seehafenbetriebe

Query language

By default, WFS requests use CQL (Contextual Query Language) queries for simplicity. CQL queries only work together with GET requests. This means that when the URL is longer than 2048 characters, they fail. While POST requests are much more flexible and able to accommodate long queries, XML is really a pain to work with and I'm not confident in my approach to construct XML queries. You can control whether to send GET or POST requests by setting `options(ffm_query_language = "XML")` or `options(ffm_query_language = "CQL")`.

See Also

[wfs_poi_open documentation](#)

[wfs_poi_open MIS record](#)

Other points of interest: [bkg_airports\(\)](#), [bkg_crossings\(\)](#), [bkg_heliports\(\)](#), [bkg_kilometrage\(\)](#), [bkg_stations\(\)](#), [bkg_trauma_centers\(\)](#)

Examples

```
# Get only seaports that co-function as inland ports
ports <- bkg_seaports(typ == "See- und Binnenhafen")
germany <- bkg_admin(level = "sta", scale = "5000", gf == 9)
plot(germany$geometry)
plot(ports$geometry, add = TRUE)
```

bkg_stations

Stations and stops

Description

Retrieve data on public transport stations and stops in Germany. Stations and stops are hierarchical. This means that stations represent the structural facilities as hierarchically superior objects and stops are hierarchically inferior parts of a station (e.g., a single platform at a bus stop).

Usage

```
bkg_stations(  
  ...,  
  bbox = NULL,  
  poly = NULL,  
  predicate = "intersects",  
  filter = NULL,  
  epsg = 3035,  
  properties = NULL,  
  max = NULL  
)
```


Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

<code>bbox</code>	An sf geometry or a boundary box vector of the format <code>c(xmin, ymin, xmax, ymax)</code> . Used as a geometric filter to include only those geometries that relate to <code>bbox</code> according to the predicate specified in <code>predicate</code> . If an sf geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
<code>poly</code>	An sf geometry. Used as a geometric filter to include only those geometries that relate to <code>poly</code> according to the predicate specified in <code>predicate</code> . Coordinates are automatically transformed to EPSG:25832 (the default CRS).
<code>predicate</code>	A spatial predicate that is used to relate the output geometries with the object specified in <code>bbox</code> or <code>poly</code> . For example, if <code>predicate = "within"</code> , and <code>bbox</code> is specified, returns only those geometries that lie within <code>bbox</code> . Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within", "contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
<code>filter</code>	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through ... Use this argument to construct more complex filters. Defaults to NULL.
<code>epsg</code>	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(...)\$epsg</code> on a spatial object that you are working with. Defaults to 3035.
<code>properties</code>	Vector of columns to include in the output.
<code>max</code>	Maximum number of results to return.

Value

A dataframe with the following columns:

- name: Geographical name of the POI
- gemeinde: Municipality name
- verwaltung: Administrative association name
- kreis: District name
- regierungs: Government region name
- bundesland: Federal state name
- stop_id: Identifier of the station or stop
- parent_st: Identifier of the parent station if applicable
- verkehrsm: Vehicle used at the station, comma-separated and sorted alphabetically
- art: Hierarchical position of a station. Can be:
 - Station: A physical structure and hierarchically superior
 - Haltestelle: Part of a structure and hierarchically inferior
- tag_f_awo: Mean departures per day in a work week
- tag_f_wo: Mean departures per day in a full week

Query language

By default, WFS requests use CQL (Contextual Query Language) queries for simplicity. CQL queries only work together with GET requests. This means that when the URL is longer than 2048 characters, they fail. While POST requests are much more flexible and able to accommodate long queries, XML is really a pain to work with and I'm not confident in my approach to construct XML queries. You can control whether to send GET or POST requests by setting `options(ffm_query_language = "XML")` or `options(ffm_query_language = "CQL")`.

See Also

[wfs_poi_open documentation](#)

[wfs_poi_open MIS record](#)

Other points of interest: [bkg_airports\(\)](#), [bkg_crossings\(\)](#), [bkg_heliports\(\)](#), [bkg_kilometrage\(\)](#), [bkg_seaports\(\)](#), [bkg_trauma_centers\(\)](#)

Examples

```
# Get all long-distance train stations
bkg_stations(verkehrsm %LIKE% "%Fernzug%", art == "Station")

# Get all platforms of long-distance train stations
bkg_stations(verkehrsm %LIKE% "%Fernzug%", art == "Haltestelle")

# Get all stops with high traffic
bkg_stations(tag_f_awo > 1000, art == "Station")
```

```
# Get all bus stops with low traffic
bkg_stations(tag_f_awo < 1, verkehrsm %LIKE% "%Bus%", art == "Station")
```

bkg_trauma_centers *Trauma centers*

Description

Retrieve data on trauma centers in Germany. A trauma center is a hospital certified by the German Society for Trauma Surgery (DGU) that is able to provide maximum care for heavily injured people. Trauma centers are organized in a hierarchical regional network that ranges from local to transregional centers.

Usage

```
bkg_trauma_centers(
  ...,
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  filter = NULL,
  epsg = 3035,
  properties = NULL,
  max = NULL
)
```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
==	=	PropertyIsEqualTo
!=	<>	PropertyIsNotEqualTo
<	<	PropertyIsLessThan
>	>	PropertyIsGreaterThan
>=	>=	PropertyIsGreaterThanOrEqualTo
<=	<=	PropertyIsLessThanOrEqualTo
%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using options(`ffm_query_language = "xml"`). See also [wfs_filter](#).

<code>bbox</code>	An <code>sf</code> geometry or a boundary box vector of the format <code>c(xmin, ymin, xmax, ymax)</code> . Used as a geometric filter to include only those geometries that relate to <code>bbox</code> according to the predicate specified in <code>predicate</code> . If an <code>sf</code> geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
<code>poly</code>	An <code>sf</code> geometry. Used as a geometric filter to include only those geometries that relate to <code>poly</code> according to the predicate specified in <code>predicate</code> . Coordinates are automatically transformed to EPSG:25832 (the default CRS).
<code>predicate</code>	A spatial predicate that is used to relate the output geometries with the object specified in <code>bbox</code> or <code>poly</code> . For example, if <code>predicate = "within"</code> , and <code>bbox</code> is specified, returns only those geometries that lie within <code>bbox</code> . Can be one of <code>"equals"</code> , <code>"disjoint"</code> , <code>"intersects"</code> , <code>"touches"</code> , <code>"crosses"</code> , <code>"within"</code> , <code>"contains"</code> , <code>"overlaps"</code> , <code>"relate"</code> , <code>"dwithin"</code> , or <code>"beyond"</code> . Defaults to <code>"intersects"</code> .
<code>filter</code>	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through <code>...</code> . Use this argument to construct more complex filters. Defaults to <code>NULL</code> .
<code>epsg</code>	An EPSG code specifying a coordinate reference system of the output. If you're unsure what this means, try running <code>sf::st_crs(...)\$epsg</code> on a spatial object that you are working with. Defaults to <code>3035</code> .
<code>properties</code>	Vector of columns to include in the output.
<code>max</code>	Maximum number of results to return.

Value

A dataframe containing the following columns:

- `name`: Geographical name of the POI
- `gemeinde`: Municipality name
- `verwaltung`: Administrative association name
- `kreis`: District name
- `regierungs`: Government region name
- `bundesland`: Federal state name
- `poi_id`: Unique primary key of a point of interest
- `tz_nummer`: Unique primary key of the trauma center
- `strasse`: Street
- `hn`: House number
- `plz`: Zip code
- `ort`: Place name
- `netzwerk`: Name of the regional trauma center network

- abteilung: Name of the medical department
- typ: Type of trauma center. Can be:
 - LTZ: local trauma center
 - RTZ: regional trauma center
 - ÜTZ: transregional trauma center

Query language

By default, WFS requests use CQL (Contextual Query Language) queries for simplicity. CQL queries only work together with GET requests. This means that when the URL is longer than 2048 characters, they fail. While POST requests are much more flexible and able to accommodate long queries, XML is really a pain to work with and I'm not confident in my approach to construct XML queries. You can control whether to send GET or POST requests by setting `options(ffm_query_language = "XML")` or `options(ffm_query_language = "CQL")`.

See Also

[wfs_poi_open documentation](#)

[wfs_poi_open MIS record](#)

Other points of interest: [bkg_airports\(\)](#), [bkg_crossings\(\)](#), [bkg_heliports\(\)](#), [bkg_kilometrage\(\)](#), [bkg_seaports\(\)](#), [bkg_stations\(\)](#)

Examples

```
# Get only trauma centers specializing on orthopedics
bkg_trauma_centers(abteilung %LIKE% "%orthopäd%")
```

```
# Get only local trauma centers
bkg_trauma_centers(typ == "LTZ")
```

```
if (requireNamespace("ggplot2")) {
  library(ggplot2)
  centers <- bkg_trauma_centers()
  ggplot() +
    geom_sf(
      data = centers[centers$typ %in% "LTZ",],
      size = 1,
      color = "lightblue1"
    ) +
    geom_sf(
      data = centers[centers$typ %in% "RTZ",],
      size = 2,
      color = "lightblue3"
    ) +
    geom_sf(
      data = centers[centers$typ %in% "ÜTZ",],
      size = 3,
      color = "lightblue4"
    ) +
  theme_void()
```

```
}

```

bkg_wcs
BKG WCS

Description

Low-level interface to BKG-style web coverage services (WCS). This function is used in all high-level functions of `ffm` that depend on a WCS, e.g., `bkg_dem`.

Usage

```
bkg_wcs(
    coverage_id,
    endpoint = coverage_id,
    version = "2.0.1",
    method = NULL,
    format = "image/tiff;application=geotiff",
    epsg = 3035,
    interpolation = NULL,
    ...
)
```

Arguments

<code>coverage_id</code>	Coverage ID. When in doubt, inspect the <code>GetCapabilities</code> of the service.
<code>endpoint</code>	Endpoint to interface. Note that <code>wcs_</code> is appended and only the rest of the product name must be provided. For example, <code>wcs_dgm200_inspire</code> becomes <code>dgm200_inspire</code> . Defaults to the value of <code>coverage_id</code> .
<code>version</code>	Service version of the WCS. Defaults to <code>2.0.1</code> .
<code>method</code>	HTTP method to use for the request. GET requests provide parameters using URL queries. Filters must be provided as CQL queries. While this is less error-prone, it allows a maximum number of only 2048 characters. Especially when providing more sophisticated spatial queries, GET queries are simply not accepted by the services. In these cases it makes sense to use POST requests instead. If NULL, the method is inferred from the type of filter query provided to <code>filter</code> (either XML or CQL). If no filter is provided, the method is inferred from <code>getOption("ffm_query_language")</code> .
<code>format</code>	Content type of the output. This value heavily depends the endpoint queried. Defaults to <code>"image/tiff;application=geotiff"</code> .
<code>epsg</code>	Numeric value giving the EPSG identifier of the coordinate reference system (CRS). The EPSG code is automatically formatted in a OGC-compliant manner. Note that not all EPSG codes are supported. Inspect the <code>GetCapabilities</code> of the target service to find out which EPSG codes are available. Defaults to <code>EPSG:3035</code> .

interpolation Method used to interpolate the coverage raster. Allowed methods depend on the capabilities of the WCS.

... Further parameters passed to the WFS query. In case of POST requests, additional namespaces that may be necessary to query the WFS. Argument names are interpreted as the prefix (e.g. `xmlns:wfs`) and argument values as namespace links.

Value

A [SpatRaster](#).

Examples

```
# Boundaries can be provided using two subset arguments
bkg_wcs(
  "dgm200_inspire__EL.GridCoverage",
  endpoint = "dgm200_inspire",
  subset = "E(548282,552280)",
  subset = "N(5800943,5804942)"
)
```

bkg_wfs

BKG WFS

Description

Low-level interface to BKG-style web feature services (WFS). This function is used in all high-level functions of `ffm` that depend on a WFS, e.g., [bkg_admin](#).

`bkg_feature_types` lists all available feature types for a given endpoint.

Usage

```
bkg_wfs(
  type_name,
  endpoint = type_name,
  version = "2.0.0",
  method = NULL,
  format = "application/json",
  layer = NULL,
  epsg = 3035,
  properties = NULL,
  filter = NULL,
  server = sgx_base(),
  ...
)

bkg_feature_types(endpoint, server = sgx_base())
```

Arguments

type_name	Feature type of the WFS to retrieve. You can use <code>bkg_feature_types</code> to retrieve a list of feature type names for a given endpoint.
endpoint	Endpoint to interface. Note that <code>wfs_</code> is appended and only the rest of the product name must be provided. For example, <code>wfs_vg250</code> becomes <code>vg250</code> . Defaults to the value of <code>type_name</code> .
version	Service version of the WFS. Usually 2.0.0, but some services still use 1.0.0 or 1.1.0.
method	HTTP method to use for the request. GET requests provide parameters using URL queries. Filters must be provided as CQL queries. While this is less error-prone, it allows a maximum number of only 2048 characters. Especially when providing more sophisticated spatial queries, GET queries are simply not accepted by the services. In these cases it makes sense to use POST requests instead. If NULL, the method is inferred from the type of filter query provided to <code>filter</code> (either XML or CQL). If no filter is provided, the method is inferred from <code>getOption("ffm_query_language")</code> .
format	Content type of the output. This value heavily depends the endpoint queried. Most services allow <code>application/json</code> but some only support GML outputs. When in doubt, inspect the <code>GetCapabilities</code> of the target service. Defaults to <code>"application/json"</code> .
layer	If <code>format</code> specifies a GML output, <code>layer</code> specifies which layer from the downloaded GML file to read. Only necessary if the GML file actually contains multiple layers. Defaults to NULL.
epsg	Numeric value giving the EPSG identifier of the coordinate reference system (CRS). The EPSG code is automatically formatted in a OGC-compliant manner. Note that not all EPSG codes are supported. Inspect the <code>GetCapabilities</code> of the target service to find out which EPSG codes are available. Defaults to <code>EPSG:3035</code> .
properties	Names of columns to include in the output. Defaults to NULL (all columns).
filter	A WFS filter query (CQL or XML) created by <code>wfs_filter</code> .
server	WFS server domain to use. Defaults to the SGX spatial data center of the BKG.
...	Further parameters passed to the WFS query. In case of POST requests, additional namespaces that may be necessary to query the WFS. Argument names are interpreted as the prefix (e.g. <code>xmlns:wfs</code>) and argument values as namespace links.

Value

An sf tibble

See Also

[bkg_wcs](#) for a low-level WCS interface

[wfs_filter](#) for filter constructors

Examples

```
bkg_feature_types("vg5000_0101")

bkg_wfs(
  "vg5000_lan",
  endpoint = "vg5000_0101",
  count = 5,
  properties = "gen",
  epsg = 4326
)[-1]

# Filters are created using `wfs_filter()`
bkg_wfs(
  "vg5000_krs",
  endpoint = "vg5000_0101",
  properties = "gen",
  filter = wfs_filter(sn_l == 10)
)[-1]
```

nuts_data

German NUTS MULTIPOLYGONS

Description

Three [sf](#) dataframes containing all geometries of German NUTS1, NUTS2, and NUTS3 regions, respectively. The reference year is 2023.

These datasets can be very useful for quickly retrieving pre-loaded boundaries without download.

Usage

`bkg_nuts1`

`bkg_nuts2`

`bkg_nuts3`

Format

For the dataframe format, see [bkg_nuts](#).

An object of class `sf` (inherits from `tbl_df`, `tbl`, `data.frame`) with 38 rows and 7 columns.

An object of class `sf` (inherits from `tbl_df`, `tbl`, `data.frame`) with 400 rows and 7 columns.

Source

© BKG (2025) dl-de/by-2-0, data sources: https://sgx.geodatenzentrum.de/web_public/gdz/datenquellen/datenquellen_vg_nuts.pdf

See Also[bkg_nuts](#)Other datasets: [admin_data](#)**Examples**

bkg_nuts1

`wfs_filter`*WFS filters*

Description

Utility functions to construct XML or CQL queries. These functions are the backend of the `filter` argument in the filter capabilities of all `ffm` functions that interact with a WFS (e.g., [bkg_admin](#), [bkg_clc](#) or [bkb_geonames](#)).

Usage

```
wfs_filter(
  ...,
  filter = NULL,
  bbox = NULL,
  poly = NULL,
  predicate = "intersects",
  geom_property = "geom",
  default_crs = 25832,
  lang = NULL
)
```

Arguments

... Used to construct CQL filters. Dot arguments accept an R-like syntax that is converted to CQL queries internally. These queries basically consist of a property name on the left, an arbitrary vector on the right, and an operator that links both sides. If multiple queries are provided, they will be chained with AND. The following operators and their respective equivalents in CQL and XML are supported:

R	CQL	XML
<code>==</code>	<code>=</code>	<code>PropertyIsEqualTo</code>
<code>!=</code>	<code><></code>	<code>PropertyIsNotEqualTo</code>
<code><</code>	<code><</code>	<code>PropertyIsLessThan</code>
<code>></code>	<code>></code>	<code>PropertyIsGreaterThan</code>
<code>>=</code>	<code>>=</code>	<code>PropertyIsGreaterThanOrEqualTo</code>
<code><=</code>	<code><=</code>	<code>PropertyIsLessThanOrEqualTo</code>

%LIKE%	LIKE	PropertyIsLike
%ILIKE%	ILIKE	
%in%	IN	PropertyIsEqualTo and Or

To construct more complex queries, you can use the `filter` argument to pass CQL queries directly. Also note that you can switch between CQL and XML queries using `options(ffm_query_language = "xml")`. See also [wfs_filter](#).

<code>filter</code>	A character string containing a valid CQL or XML filter. This string is appended to the query constructed through Use this argument to construct more complex filters. Defaults to NULL.
<code>bbox</code>	An sf geometry or a boundary box vector of the format <code>c(xmin, ymin, xmax, ymax)</code> . Used as a geometric filter to include only those geometries that relate to <code>bbox</code> according to the predicate specified in <code>predicate</code> . If an sf geometry is provided, coordinates are automatically transformed to EPSG:25832 (the default CRS), otherwise they are expected to be in EPSG:25832.
<code>poly</code>	An sf geometry. Used as a geometric filter to include only those geometries that relate to <code>poly</code> according to the predicate specified in <code>predicate</code> . Coordinates are automatically transformed to EPSG:25832 (the default CRS).
<code>predicate</code>	A spatial predicate that is used to relate the output geometries with the object specified in <code>bbox</code> or <code>poly</code> . For example, if <code>predicate = "within"</code> , and <code>bbox</code> is specified, returns only those geometries that lie within <code>bbox</code> . Can be one of "equals", "disjoint", "intersects", "touches", "crosses", "within", "contains", "overlaps", "relate", "dwithin", or "beyond". Defaults to "intersects".
<code>geom_property</code>	Name of the geometry property included in the WFS. In most cases, this is "geom", but there are some exceptions.
<code>default_crs</code>	A WFS defines a default CRS in which coordinates for spatial filtering have to be provided. For BKG services, this is usually EPSG:25832. All sf objects provided through <code>bbox</code> or <code>poly</code> are first transformed to this CRS before creating the query.
<code>lang</code>	Query language to use for constructing the query. One of "cql" and "xml". By default, almost all ffm functions use CQL because it is simpler and less prone to errors. However, CQL is limited in terms of query size. Especially when providing a <code>poly</code> , URLs can become so long that the WFS server will decline them. XML can be a valid alternative to construct large queries. Additionally, some services like the one used by bkg_geonames only support XML. If NULL, defaults to <code>getOption("ffm_query_language")</code> .

Value

A CQL query or an XML query depending on the `lang` argument.

Examples

```
# CQL and XML support mostly the same things
wfs_filter(ags %LIKE% "05%", lang = "cql")
wfs_filter(ags %LIKE% "05%", lang = "xml")
```

```
bbox <- c(xmin = 5, ymin = 50, xmax = 7, ymax = 52)
wfs_filter(bbox = bbox, lang = "cql")
wfs_filter(bbox = bbox, lang = "xml")

# Using `filter`, more complex queries can be built
wfs_filter(ars %LIKE% "%0", filter = "regierungs NOT IS NULL")

wfs_filter(
  filter = "<fes:Not>
    <fes:PropertyIsNull>
      <fes:ValueReference>aussprache</fes:ValueReference>
    </fes:PropertyIsNull>
  </fes:Not>",
  lang = "xml"
)
```

Index

- * **datasets**
 - admin_data, 2
 - nuts_data, 57
- * **non-administrative regions**
 - bkg_area_codes, 14
 - bkg_authorities, 16
 - bkg_grid, 30
 - bkg_kfz, 35
 - bkg_ror, 43
- * **points of interest**
 - bkg_airports, 11
 - bkg_crossings, 20
 - bkg_heliports, 32
 - bkg_kilometrage, 38
 - bkg_seaports, 46
 - bkg_stations, 48
 - bkg_trauma_centers, 51
- admin_data, 2, 8, 42, 58
- bkg_admin, 3, 3, 9–11, 15, 29, 37, 42, 55, 58
- bkg_admin_archive (bkg_admin), 3
- bkg_admin_hierarchy, 8, 8
- bkg_admin_highres (bkg_admin), 3
- bkg_ags, 10, 27–29
- bkg_airports, 11, 22, 35, 40, 48, 50, 53
- bkg_amr (bkg_ror), 43
- bkg_area_codes, 14, 18, 28, 32, 37, 46
- bkg_ars, 28, 29
- bkg_ars (bkg_ags), 10
- bkg_authorities, 8, 15, 16, 32, 37, 46
- bkg_bkr (bkg_ror), 43
- bkg_clc, 18, 58
- bkg_crossings, 13, 20, 35, 40, 48, 50, 53
- bkg_dem, 22, 54
- bkg_dlm, 23, 28, 29
- bkg_endonyms, 11, 15
- bkg_endonyms (bkg_geonames), 26
- bkg_feature_types, 28
- bkg_feature_types (bkg_wfs), 55
- bkg_geonames, 11, 15, 26, 59
- bkg_germany (admin_data), 2
- bkg_ggr (bkg_ror), 43
- bkg_grid, 8, 15, 18, 30, 37, 46
- bkg_grid_fast (bkg_grid), 30
- bkg_grid_full (bkg_grid), 30
- bkg_heliports, 13, 22, 32, 40, 48, 50, 53
- bkg_kfz, 8, 15, 18, 32, 35, 46
- bkg_kilometrage, 13, 22, 35, 38, 48, 50, 53
- bkg_kmr (bkg_ror), 43
- bkg_krg (bkg_ror), 43
- bkg_krs (admin_data), 2
- bkg_mbe (bkg_ror), 43
- bkg_mkro (bkg_ror), 43
- bkg_nuts, 8, 40, 57, 58
- bkg_nuts1 (nuts_data), 57
- bkg_nuts2 (nuts_data), 57
- bkg_nuts3 (nuts_data), 57
- bkg_quasigeoid, 42
- bkg_rg (bkg_ror), 43
- bkg_ror, 8, 15, 18, 32, 37, 43
- bkg_seaports, 13, 22, 35, 40, 46, 50, 53
- bkg_states (admin_data), 2
- bkg_stations, 13, 22, 35, 40, 48, 48, 53
- bkg_trauma_centers, 13, 22, 35, 40, 48, 50, 51
- bkg_wcs, 54, 56
- bkg_wfs, 28, 55
- nuts_data, 3, 8, 42, 57
- req_timeout, 5, 9, 31, 41, 43, 45
- sf, 2, 57
- SpatRaster, 23, 43, 55
- wfs_filter, 4, 10–12, 14, 15, 17, 19, 21, 24, 27, 29, 33, 36, 39, 46, 49, 52, 56, 58, 59