

Package: kwb.geoportal (via r-universe)

October 11, 2026

Title R Package for Getting Spatial Data from Berlin Geoportal

Version 0.0.0.9000

Description R Package for getting spatial data from Berlin Geoportal
(<https://gdi.berlin.de/geonetwork/srv/ger/catalog.search#/search>).

License MIT + file LICENSE

URL <https://github.com/KWB-R/kwb.geoportal>

BugReports <https://github.com/KWB-R/kwb.geoportal/issues>

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports dplyr, purrr, sf, stringr, tibble, xml2

Suggests covr, DT, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libicu-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository <https://kwb-r.r-universe.dev>

Date/Publication 2026-09-11 12:18:37 UTC

RemoteUrl <https://github.com/KWB-R/kwb.geoportal>

RemoteRef HEAD

RemoteSha a004cdcc68dc8a6006e2497c1fc51b326d01fdc2

Contents

find_wfs	2
list_wfs_layers	3
parse_gn_link	4
read_metadata	5
read_metadata_all	6

read_wfs	7
read_wfs_sf_auto	8
wfs_base_url	9
wfs_service_name	10

Index	11
--------------	-----------

find_wfs	<i>Find WFS services in the Geoportal catalogue</i>
----------	---

Description

Searches the GeoNetwork catalogue read by `read_metadata_all()` for records that offer a download service, and returns one row per matching WFS link. The service column of the result is what `read_wfs()` takes as its first argument.

Usage

```
find_wfs(pattern = NULL, metadata = NULL, protocol = "OGC:WFS", ...)
```

Arguments

pattern	regular expression matched, case-insensitively, against the record title, its abstract and the link URL. NULL (default) returns every WFS link.
metadata	result of a previous <code>read_metadata_all()</code> call. Pass this to search repeatedly without downloading the catalogue again. If NULL (default) the catalogue is read.
protocol	link protocol to keep (default: "OGC:WFS"). Pass "OGC:WMS" to find map services instead, or NULL for all of them.
...	further arguments passed to <code>read_metadata_all()</code> , e.g. base_url.

Details

This answers "which endpoint belongs to dataset X". It is a convenience for discovery: once the service name is known it should be written down in the calling code, so that fetching the data no longer depends on the catalogue being reachable.

Value

tibble with columns title, service, link_url, link_desc, link_protocol, geonet_uuid, sorted by title. Warns, and returns no rows, when metadata holds no links at all or none of them uses protocol – both of which are a different thing from pattern not matching. The second warning names the protocols that are on offer.

Examples

```
## Not run:
catalogue <- read_metadata_all()
find_wfs("kanalisation", metadata = catalogue)
find_wfs("alkis", metadata = catalogue)

## End(Not run)
```

list_wfs_layers*List the feature types offered by a WFS service*

Description

Reads the service's GetCapabilities document and returns one row per feature type. Use this to find out which layer to pass to [read_wfs\(\)](#) – it is the authoritative source, unlike the catalogue, which describes datasets rather than layers.

Usage

```
list_wfs_layers(
  service,
  version = c("2.0.0", "1.1.0", "1.0.0"),
  host = "https://gdi.berlin.de/services/wfs"
)
```

Arguments

service	service name, see wfs_base_url() , or a full endpoint URL.
version	WFS version, one of "2.0.0" (default), "1.1.0", "1.0.0".
host	see wfs_base_url() .

Value

tibble with columns typename, title, abstract, crs.

Examples

```
## Not run:
list_wfs_layers("wsg")

## End(Not run)
```

parse_gn_link	<i>Parse a single GeoNetwork link element</i>
---------------	---

Description

GeoNetwork encodes links as a single string separated by |, e.g.: |Darstellungsdienst (WMS)|https://...|OGC:WMS|||
This helper splits such a string into named columns.

Usage

```
parse_gn_link(x)
```

Arguments

x	Character string as found inside a <link> XML node.
---	---

Details

The fields are ordered name, description, URL, protocol, MIME type, order. The URL is the one field that can be recognised on its own, so it is located first and the remaining fields are read relative to it; that keeps the columns aligned for a catalogue that pads the record differently.

Note that a parsed `link_protocol` is only as good as the catalogue: the GDI Berlin records repeat the description there instead of naming a protocol, so filtering on it needs `gn_link_protocol()`.

Value

A one-row tibble with columns:

- link_name
- link_desc
- link_url
- link_protocol
- link_mime
- link_order

Examples

```
parse_gn_link("|Darstellungsdienst (WMS)|https://example.org/wms?|OGC:WMS|||")
```

read_metadata	<i>Read GeoNetwork service metadata (one row per service)</i>
---------------	---

Description

This function reads a GeoNetwork XML search response (as delivered by <https://gdi.berlin.de/geonetwork/srv/ger/>) and converts it into a tidy tibble with **one row per metadata record**. All `<link>` elements of a record are kept together in a **list column** called `links`, where each entry is itself a tibble created by `parse_gn_link()`.

Usage

```
read_metadata(path_xml)
```

Arguments

<code>path_xml</code>	Path or URL to the GeoNetwork XML document. This can be a local file (e.g. "geoportal_metadaten.xml") or a remote URL such as "https://gdi.berlin.de/geonetwork/srv/ger/".
-----------------------	--

Details

This structure is convenient when you want to keep the dataset-level information (title, abstract, uuid, ...) together, but still be able to inspect or unnest all service/download/view links later on.

The function assumes a GeoNetwork-style XML with `<metadata>` elements and the namespace `geonet:` available for the info block. It is tailored to the GDI Berlin instance but should work for other similar GeoNetwork responses that use the same link encoding (| separated).

If a record has **no** `<link>` elements, the `links` column will contain a single-row tibble with all NA values. This preserves the 1:1 alignment between records and rows.

Value

A tibble with one row per metadata record and the columns:

- geonet_uuid** UUID from `<geonet:info><uuid>` (character).
- geonet_id** Internal GeoNetwork id from `<geonet:info><id>` (character).
- title** Dataset/service title.
- abstract** Dataset/service abstract/description.
- serviceType** Service type, if present (e.g. WMS).
- types** Semicolon-separated `<type>` elements.
- source_logo** Logo path, if present.
- links** List column; each element is a tibble with the parsed links.

Examples

```
## Not run:
df <- read_geonetwork_services(
  "https://gdi.berlin.de/geonetwork/srv/ger/q?facet.q=type/service&resultType=details&sortBy=changeDate&from=1&to=1000"
)
dplyr::glimpse(df)
df$links[[1]]

## End(Not run)
```

read_metadata_all	<i>Read all GeoNetwork / Geoportal metadata in chunks</i>
-------------------	---

Description

This function uses [read_metadata\(\)](#) repeatedly to fetch **all** available metadata records from a GeoNetwork endpoint that supports from / to pagination (like the GDI Berlin instance).

Usage

```
read_metadata_all(
  base_url =
    "https://gdi.berlin.de/geonetwork/srv/ger/q?facet.q=type/service&resultType=details&sortBy=changeDate&from=1&to=1000"
  chunk_size = 100
)
```

Arguments

base_url	Base GeoNetwork query URL without from and to parameters. Must return an XML with a <summary count="..."> node. Defaults to the GDI Berlin service search.
chunk_size	Number of records per request. Default: 100.

Details

It first downloads the initial XML, reads the <summary count="..."> attribute to know how many records exist, then iterates in chunks (default: 100) until all records are read.

Value

A tibble with one row per metadata record, identical in structure to the return value of [read_metadata\(\)](#), but for **all** pages.

See Also

[read_metadata\(\)](#)

Examples

```
## Not run:
all_md <- read_metadata_all()
nrow(all_md)

## End(Not run)
```

read_wfs

Read a WFS layer from the Berlin Geoportal

Description

Downloads one feature type of a WFS service and reads it into an `sf` object. This is the data counterpart to [read_metadata\(\)](#), which only describes where a dataset lives.

Usage

```
read_wfs(
  service,
  layer = NULL,
  srs = "EPSG:25833",
  version = c("2.0.0", "1.1.0", "1.0.0"),
  output_format = "application/json",
  count = NULL,
  start_index = NULL,
  host = "https://gdi.berlin.de/services/wfs",
  quiet = FALSE
)
```

Arguments

<code>service</code>	service name, see wfs_base_url() , or a full endpoint URL.
<code>layer</code>	feature type to request. Either the fully qualified name (e.g. "wsg:wsg") or the bare layer name, in which case the service name is used as the namespace prefix. If <code>NULL</code> (default) the service is asked for its feature types via list_wfs_layers() ; this only succeeds when the service offers exactly one.
<code>srs</code>	coordinate reference system of the response (default: "EPSG:25833", the ETRS89 / UTM 33N used by the Berlin services).
<code>version</code>	WFS version, one of "2.0.0" (default), "1.1.0", "1.0.0".
<code>output_format</code>	value of OUTPUTFORMAT (default: "application/json", i.e. GeoJSON). Pass <code>NULL</code> to let the server choose, which usually yields GML.
<code>count</code>	maximum number of features to return, or <code>NULL</code> (default) for all of them. Sent as COUNT for WFS 2.0.0 and as MAXFEATURES otherwise.

start_index	index of the first feature to return, or NULL (default). Together with count this allows paging through layers that exceed the server's response limit. WFS 2.0.0 only.
host	see <code>wfs_base_url()</code> .
quiet	suppress the progress messages of <code>sf::read_sf()</code> and of this function (default: FALSE).

Details

The request is a plain GetFeature call against <host>/<service>?SERVICE=WFS&VERSION=...&REQUEST=GetFeature&TYPE=... which is the form the Berlin endpoints answer.

Value

an sf data frame.

Examples

```
## Not run:
# Water protection zones, the layer the model's dashboard uses
wsg <- read_wfs("wsg", "wsg")

# ATKIS river centrelines
rivers <- read_wfs("atkis", "b17_ax_gewaesserstationierungsachse_1")

## End(Not run)
```

read_wfs_sf_auto	<i>Read a WFS layer (auto-detect from URL)</i>
------------------	--

Description

Simplified helper to read a single WFS layer into R using `sf`. The function automatically extracts the layer ID from the WFS URL, assuming the URL pattern .../wfs/<layer>?REQUEST=GetCapabilities....

Usage

```
read_wfs_sf_auto(url, crs_target = 25833, bbox = NULL, quiet = FALSE)
```

Arguments

url	character(1). Full WFS URL (e.g. a GetCapabilities link).
crs_target	integer(1) or NULL. EPSG code to reproject the result to. Defaults to 25833 (ETRS89 / UTM zone 33N). Use NULL to keep native CRS.
bbox	numeric(4) or NULL. Optional bounding box (c(xmin, ymin, xmax, ymax)), passed to the GetFeature request.
quiet	logical(1). Suppress download messages (default: FALSE).

Value

An sf object with the requested WFS layer.

Examples

```
## Not run:
# Read Berlin ALKIS Bezirke directly from GetCapabilities URL
bezirke <- read_wfs_sf_auto(
  "https://gdi.berlin.de/services/wfs/alkis_bezirke?REQUEST=GetCapabilities&SERVICE=wfs"
)
plot(bezirke["bezirk"])

## End(Not run)
```

wfs_base_url

Base URL of a Berlin Geoportal WFS service

Description

The Berlin spatial data infrastructure publishes one WFS endpoint per dataset under a common path, e.g. `https://gdi.berlin.de/services/wfs/atkis` for the ATKIS layers. This helper only composes that URL; it does not contact the server.

Usage

```
wfs_base_url(service, host = "https://gdi.berlin.de/services/wfs")
```

Arguments

service	name of the service, i.e. the last path element of the WFS endpoint (e.g. "atkis", "wsg"). This is also the namespace prefix of the feature types offered by that service.
host	base path of the WFS endpoints (default: "https://gdi.berlin.de/services/wfs").

Value

character of length one, the service URL without query string. Signals an error when service is not a single path element, because the resulting URL would otherwise be handed to `xml2::read_xml()`, which parses any string containing < or > as literal XML rather than fetching it.

Examples

```
wfs_base_url("atkis")
```

wfs_service_name	<i>Service name of a WFS endpoint URL</i>
------------------	---

Description

Extracts the last path element of an endpoint URL, i.e. the name that `read_wfs()` and `list_wfs_layers()` expect.

Usage

```
wfs_service_name(url)
```

Arguments

`url` endpoint URL, with or without query string.

Value

character vector of service names, NA where none could be found.

Examples

```
wfs_service_name("https://gdi.berlin.de/services/wfs/atkis?SERVICE=WFS")
```

Index

`find_wfs`, [2](#)

`gn_link_protocol()`, [4](#)

`list_wfs_layers`, [3](#)
`list_wfs_layers()`, [7](#), [10](#)

`parse_gn_link`, [4](#)
`parse_gn_link()`, [5](#)

`read_metadata`, [5](#)
`read_metadata()`, [6](#), [7](#)
`read_metadata_all`, [6](#)
`read_metadata_all()`, [2](#)
`read_wfs`, [7](#)
`read_wfs()`, [2](#), [3](#), [10](#)
`read_wfs_sf_auto`, [8](#)

`sf::read_sf()`, [8](#)

`wfs_base_url`, [9](#)
`wfs_base_url()`, [3](#), [7](#), [8](#)
`wfs_service_name`, [10](#)