Package: gsdata (via r-universe)

June 7, 2026

Type Package

Title Interface to the GeoSphere Austria DataHub API (Data Access)

Version 0.0-8

Date 2025-12-05

Maintainer Reto Stauffer <reto.stauffer@uibk.ac.at>

Depends httr, zoo, sf, parsedate, dplyr

Suggests utils, ncdf4, stars, knitr, rmarkdown, tinytest

Description Package to download data from the GeoSphere data hub (Austrian National Weather Service) which is the data provider. Currently solely allows to download station data (no gridded data). More details about the data sets can be found on <https://data.hub.geosphere.at/> as well as in the API documentation available via <https://dataset.api.hub.geosphere.at/v1/docs/>.

URL https://retostauffer.github.io/gsdata

License GPL-2

VignetteBuilder knitr

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

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

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

Date/Publication 2026-01-19 19:47:00 UTC

RemoteUrl https://github.com/retostauffer/gsdata

RemoteRef HEAD

RemoteSha d811436d9b98ca83e1c0ee9770d0fb9ccd934d2c

Contents

`API_GET` *Handling API Calls*

Description

Small helper function to handle http requests to the API.

Usage

```
API_GET(
  URL,
  config = NULL,
  query = NULL,
  expected_class = NULL,
  verbose = FALSE
)
```

Arguments

`URL`	the URL to be called.
`config`	`NULL` or `list`, forwarded to `httr::GET`.
`query`	`NULL` or `list`, forwarded to `httr::GET`.
`expected_class`	`NULL` or character vector. If set, it is checked if the returned object is of this class. If not, a warning will be thrown (no error).
`verbose`	logical, shows some additional information if `TRUE`.

Value

Returns the object we get from `httr::content()` after a successful API call. If an error is detected, an error with additional details will be displayed.

Author(s)

Reto Stauffer

gs_baseurl *Getting API Base URL*

Description

Just returns the base URL for API requests

Usage

```
gs_baseurl(version = 1L)
```

Arguments

version	integer, defaults to 1.

Value

String, base URL for the API.

Author(s)

Reto Stauffer

gs_datasets *Getting Available Datasets*

Description

The GeoSphere Austria (formerly ZAMG) datahub API provides an endpoint to get available datasets. This function returns a (possibly pre-filtered) `data.frame` containing the dataset `type`, `mode` and `resource_id` needed to perform the data requests (see e.g., `gs_stationdata()`) amongst some additional information.

Usage

```
gs_datasets(
  type = "station",
  mode = NULL,
  version = 1L,
  config = list(),
  verbose = FALSE
)
```

Arguments

type	NULL or character of length 1 to filter the request.
mode	NULL or character of length 1 to filter the request. Currently defaults to mode = "station".
version	integer, API version (defaults to 1).
config	empty list by default; can be a named list to be fowrarded to the httr::GET request if needed.
verbose	logical, if set TRUE some more output will be produced.

Details

The API provides an enpoint to get all available data sets which can be filtered using the arguments type and/or mode. Classical usecase:

Return all data sets where mode == "historical":

- gs_datasets(mode = "historical")

Return all data sets where type == "grid":

- gs_datasets(type = "grid")

Can be combined (setting both type and mode).

Value

Returns a data.frame with all available data types and API endpoints. When both type and mode are equal to NULL all data available via the API will be returned. The most important information of this return is the type, mode, as well as resource_id which is used to perform data requests.

Author(s)

Reto Stauffer

See Also

gs_stationdata

---

gs_metadata *Downloading Dataset Meta Data*

Description

Downloading Dataset Meta Data

Usage

```
gs_metadata(
  mode,
  resource_id,
  type = NULL,
  version = 1L,
  config = list(),
  expert = FALSE,
  verbose = FALSE
)
```

Arguments

mode	character, specify mode of data.
resource_id	character, specify resource identifier of data.
type	NULL or character. Only required if a data set is available in more than one type (e.g., \"grid\" and \"timeseries\").
version	integer, API version (defaults to 1L).
config	empty list by default; can be a named list to be fowrarded to the httr::GET request if needed.
expert	logical, defaults to FALSE. If TRUE the script will not check if the input arguments are valid. May result in unsuccessful requests but increases the speed as gs_datasets() does not have to be called (one less API request).
verbose	logical, if set TRUE some more output will be produced.

Value

Named list with a series of information about the dataset. Most importantly this function returns information about the stations for which data is available as well as what parameters are available. Note that the availability of data depends on the station; the meta information only provides an overview of what is possibliy avialable.

- $stations: an sf object (spatial feature data frame) containing information of all stations belonging to this dataset including geographical location, name, and id (the station identifier) wihch is used when retrieving data (see e.g., gs_stationdata()).
- $parameters: a data.frame containing the name of the parameters used when retrieving the data (see e.g., gs_stationdata()) as well as a parameter and description. Only available in German, tough.

In addition, the following information will be returned as separate entries in the list:

- $title/$id_type: title/id type of the dataset
- $frequency: observation frequency/temporal interval (see also gs_temporal_interval())
- $type: data type (e.g., "station")
- $mode: data set mode (e.g., "historical")
- $response_formats: formats the API provides
- $start_time/$end_time: date/time range of availability of this data set
- $url: URL; origin of the data set

Author(s)

Reto Stauffer

Examples

```
## Loading meta information for data set with
## mode == "historical" and resource_id = "tawes-v1-10min"
tawes <- gs_metadata("historical", "tawes-v1-10min")

## Uses partial matching, thus this short form can be used in case
## there is only one match (one specific data set). With verbose = TRUE
## a message will tell which meta data set will be requested.
synop <-  gs_metadata("hist", "synop", verbose = TRUE)

## generic sf plotting; variable 'altitude'
plot(synop$stations["altitude"], pch = 19, cex = 2)
```

---

gs_stationdata *Downloading Station Data*

Description

Accessing the API endpoint v<version>/station, see https://dataset.api.hub.geosphere.at/v1/docs/getting-started.html.

Usage

```
gs_stationdata(
  mode,
  resource_id,
  parameters = NULL,
  start = NULL,
  end = NULL,
  station_ids,
  expert = FALSE,
  version = 1L,
  drop = TRUE,
  verbose = FALSE,
  format = NULL,
  limit = 2e+05,
  config = list()
)
```

Arguments

mode	character, specify mode of data.
resource_id	character, specify resource identifier of data.
parameters	character vector to define which parameters to process.
start, end	object of class Date, POSIXt, or character. In case of character in a non-ISO format format can be used (see below). Not needed (ignored) when mode = "current".
station_ids	integer vector with the station IDs to be processed.
expert	logical, defaults to FALSE. If TRUE the script will not check if the input arguments are valid. May result in unsuccessful requests but increases the speed as gs_datasets() and gs_metadata() do not have to be called (two API requests less).
version	integer, API version (defaults to 1L).
drop	logical, if TRUE parameters and times with no data are removed before returning the data.
verbose	logical, if set TRUE some more output will be produced.
format	NULL (default) or character string, used if start/end are characters in a specific (non ISO) format.
limit	integer, API data request limit. If the request sent by the user exceeds this limit, the request will be split into batches automatically. Set to 2e5 as the limit stated on the API documentation (1e6) will not be accepted.
config	empty list by default; can be a named list to be fowrarded to the httr::GET request if needed.

Details

This function is a convenience function for downloading different sets of station data from the GeoSphere Austria data hub (formerly ZAMG). The API may change and additional resources may be added, for details see https://dataset.api.hub.geosphere.at/v1/docs/user-guide/endpoints.html.

To see what's available call gs_datasets("station").

The API has a limit for the number of elements for one request. The calculation is based on the number of expecte elements (i.e., number of stations times number of parameters times number of time steps). This function will pre-calculate the number of expected elements and split the request into batches along the time dimension - if needed.

Value

If only data for one single station (length(station_ids) == 1) is requested, a zoo object will be returned if data is available. If no data is available, NULL will be returned.

When multiple stations are requested a list of zoo object (or NULL if no data is available) is returned. The name of the list corresponds to the station id requested.

Author(s)

Reto Stauffer

Examples

```
########################################################################
## Latest observations for two tawes stations in Innsbruck.
## Parameters TL (air temperature 2m above ground), TS (air temperature 5cm
## above ground) and RR (amount of rain past 10 minutes).
innsbruck <- gs_stationdata(mode        = "current",
                            resource_id = "tawes-v1-10min",
                            parameters  = c("TL", "TS", "RR"),
                            station_ids = c(11121, 11320),
                            expert      = TRUE)
# Air temp
sapply(innsbruck, function(x) x$TL)
# Precipitation (rain)
sapply(innsbruck, function(x) x$RR)

## Not run:
########################################################################
## Example for synop data

## Loading meta information
meta <- gs_metadata(mode = "historical", resource_id = "synop-v1-1h")
## For station information check
head(meta$stations)
## For available parameters (for this mode/resource_id) check
head(meta$parameters)

## Getting data over 48 hours for one single station
## Note: If expert = FALSE (default) gs_stationdata()
## will internally call gs_metadata() once more to check
## if the requested station_ids as well as the parameters
## exist for the data set specified (mode/resource_id).
mayrhofen <- gs_stationdata(mode        = "historical",
                            resource_id = "synop-v1-1h",
                            start       = "2020-01-01",
                            end         = "2020-01-03",
                            parameters  = c("T", "Td", "ff"),
                            station_ids = 11330, verbose = TRUE)

library("zoo")
plot(mayrhofen, screen = c(1, 1, 2), col = c(2, 3, 4))

## Getting data over 48 hours for three stations simultanously
## Mayrhofen Tirol, Achenkirch Tirol (no data), and Innsbruck Airport Tirol
x <- gs_stationdata(mode        = "historical",
                    resource_id = "synop-v1-1h",
                    start       = "2020-01-01",
                    end         = "2020-01-03",
                    parameters  = c("T", "Td", "ff"),
```

```
                    station_ids = c(11330, 11328, 11120),
                    expert      = TRUE)
plot(x[["11330"]], screen = c(1, 1, 2), col = c(2, 3, 4))
is.null(x[["11328"]])
plot(x[["11120"]], screen = c(1, 1, 2), col = c(2, 3, 4))

######################################################################
## Example for daily climatological records
meta <- gs_metadata("historical", "klima-v2-1d")
achenkirch <- gs_stationdata(mode        = "historical",
                             resource_id = "klima-v2-1d",
                             start       = "2020-06-01",
                             end         = "2022-12-31",
                             parameters  = c("rr", "rr_i", "rr_iii", "so_h"),
                             station_ids = 8807,
                             expert      = TRUE)
head(achenkirch)
plot(achenkirch, type = "h")


######################################################################
## Example for 10min KLIMA data
# meta$parameter contains available parameters,
# meta$stations  available stations
meta <- gs_metadata("historical", "klima-v2-10min")
uibk <-  gs_stationdata(mode        = "historical",
                        resource_id = "klima-v2-10min",
                        start       = "2010-11-01",
                        end         = "2011-02-01",
                        parameters  = c("tl", "ffam", "ffx"),
                        station_ids = 11803,
                        expert      = TRUE)
plot(uibk,
     screens = c(1, 2, 2),
     col = c(2, 4, 8),
     ylab = c("temperature", "mean wind\nand gusts"))


######################################################################
## Example for 10min TAWES data
## NOTE/WARNING:
##   ! "tawes" is not quality controlled and provides limited
##   ! amount of data. Consider to use the "klima-v2-10min" data set which
##   ! provides long-term historical data for the same stations with the
##   ! same temporal resolution, however, the station IDs and
##   ! parameter names (as well as avilavle parameters) will differ
##   ! (check meta data).
# meta$parameter contains available parameters,
# meta$stations  available stations
meta <- gs_metadata("historical", "tawes-v1-10min")
uibk <- gs_stationdata(mode         = "historical",
                       resource_id  = "tawes-v1-10min",
                       start        = Sys.Date() - 30,
```

```
                        end          = Sys.Date(),
                        parameters   = c("TL", "TP", "FFAM", "FFX"),
                        station_ids  = 11320,
                        expert       = TRUE)
plot(uibk,
     screens = c(1, 1, 2, 2),
     col = c(2, 3, 4, 8),
     ylab = c("temperature\nand dewpoint", "mean wind\nand gusts"))


######################################################################
## Example for annual histalp data
gs_metadata("historical", "histalp-v1-1y")
bregenz <- gs_stationdata(mode        = "historical",
                          resource_id = "histalp-v1-1y",
                          start       = "1854-01-01",
                          end         = "2022-01-01",
                          parameters  = c("R01", "T01"),
                          station_ids = 23,
                          expert      = TRUE)
plot(bregenz, col = c(4, 2))

## End(Not run)
```

`gs_temporal_interval` *Extract/Calculate Temporal Interval*

Description

Helper function to extract and return the temporal interval (in seconds) for different data sets based on the `resource_id` identifier. Used to estimate the number of expected values to be retrieved as there is per-request limit.

Usage

```
gs_temporal_interval(resource_id)
```

Arguments

`resource_id`	character, name of the `resource_id`

Value

Returns an integer vector of the same length as the input vector `resource_id` with the temporal resolution of the data set in seconds or `NA` in case the string could not have been decoded.

Author(s)

Reto Stauffer

Examples

```
gs_temporal_interval("test-1d-string")
ds <- gs_datasets()
gs_temporal_interval(ds$resource_id)
```

---

gsdata *gsdata: Interface to the GeoSphere Austria DataHub API (Data Access)*

Description

This package allows convenient access to data provided by GeoSphere Austria (Austrias federal agency for geology, geophysics, climatology and meteorology) via their data API which exists since around mid 2023.

Details

The API not only provides access to station data (the one thing currently covered by this package; will be extended) but also access to spatial data; a catalogue which has been extended over and over again over the past 10 months. Details about all available data sets and their temporal and spatial extent can be found on their website:

- https://data.hub.geosphere.at/

Data request limit

The API has a request limit; a limit to how much data one is allowed to retrieve in one API request. Details on the current limit can be found in the GeoSphere Dataset API Documentation.

This package internally tries to estimate the request size and split the request into multiple batches in case one single request would (likely) exceed these limits.

Thus, one single call to e.g., `gs_stationdata()` can trigger multiple API calls. If used without `expert = TRUE` two initial calls are made to check if the data set requested does exist, and that the stations and parameters requested exist in this data set. If the data request needs to be split in addition, this can cause a series of calls to the API which also has a limit on number of requests per time.

In the worst case this causes a temporary ban (timeout due to too many requests) from the servers. One way around is to limit the number of requests per time, more details about this in the next section.

Cooldown time/limiting number of requests per time

Note that each function call can result in multiple API requests which can lead to a timeout (too many requests). To avoid running into timeout issues:

- use `expert = TURUE` where possible as it lowers the number of calls to the api.

- request data for multiple stations at once, especially when requesting short time periods/few parameters as, in the best case, all data can be retrieved on one single call (if below estimated data request limit).
- wait between requests using e.g., `Sys.sleep(...)`.
- or use the packages own 'cooldown' option. By default, a cooldown time of `0.1` seconds is used (the minimum time between two requests. You can set a custom cooldown time via `options('gsdata.cooldown' = 1)`. Will overwrite the default and ensure that there will be at least one second between consecutive API calls. If you have no time critical requests this is a good way to be nice to the data provider!

Author(s)

Maintainer: Reto Stauffer <reto.stauffer@uibk.ac.at> (ORCID)

See Also

Useful links:

- https://retostauffer.github.io/gsdata

`show_http_status_and_terminate`

Show HTTP Error Status and Terminate

Description

This function is called whenever `httr::GET` returns an http status code out of the `200` range (success). Shows `http_status` code information alongside with additional messages returned by the API (if any).

Usage

```
show_http_status_and_terminate(scode, xtra = NULL)
```

Arguments

`scode`	numeric, http status code.
`xtra`	`NULL` or named list with additional information.

Value

No return, will terminate R.

Author(s)

Reto Stauffer

Index