Package 'dawaR'

August 27, 2026

Title An API Wrapper for 'DAWA' - 'The Danish Address Web API'

Date 2026-08-26

Version 0.3.3

Description Functions for interacting with all sections of the official 'Danish Address Web API' (also known as 'DAWA') <https://api.dataforsyningen.dk>. The development of this package is completely independent from the government agency, Klimadatastyrelsen, who maintains the API.

License GPL (>= 3)

Encoding UTF-8

URL https://dawar.aleksanderbl.dk/, https://github.com/aleksanderbl29/dawaR

BugReports https://github.com/aleksanderbl29/dawaR/issues

Suggests knitr, rmarkdown, ggplot2, testthat (>= 3.0.0), cowplot (>= 1.1.0), rmapshaper (>= 0.5.0), microbenchmark (>= 1.5.0), pkgdown, dplyr, devtools, vcr (>= 0.6.0)

Config/testthat/edition 3

Config/testthat/parallel true

Depends R (>= 4.1.0)

Imports memoise (>= 2.0.0), cli, httr2, sf, tidyRSS, rlang, curl, utils

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Aleksander Bang-Larsen [aut, cre, cph] (ORCID: <https://orcid.org/0009-0007-7984-4650>), Agency of Climate Data [ctb] (The Danish Agency of Climate Data provides this API for the public and have not actively contributed to the development nor distribution in this package.)

Maintainer Aleksander Bang-Larsen <contact@aleksanderbl.dk>

Repository CRAN

Date/Publication 2026-08-27 12:00:26 UTC

Contents

autocomplete 2
available_sections 3
connection_check 3
dawa 4
get_data 5
get_map_data 6
reverse 7
status_check 8

Index **9**

`autocomplete` *Base function for interacting with* `autocomplete` *of sections.*

Description

This function is very handy when needing to either validate or simply "fill in gaps" with autocompletion from the API.

Usage

```
autocomplete(section, input, ...)
```

Arguments

`section`	Defines what section of the api should be called. `available_sections()` exposes all options for this parameter.
`input`	The input to autocomplete. Could be a name, a place and similar.
`...`	<dynamic-dots> input query parameters.

Value

Returns the autocomplete information for the provided `section` given the input.

Examples

```
if (connection_check()) {
  autocomplete("regioner", "midt")
}
```

available_sections *List of available sections for* dawa()

Description

Simple function to list all available options for the section parameter in dawa(). Useful when getting to know the base api function of this package. To filter based on funciton usage you can provide format = "geojson" to ensure compatability with get_map_data() and func = "get_data" to ensure compatability with get_data().

Usage

```
available_sections(as_list = FALSE, format = NULL, verbose = TRUE, func = NULL)
```

Arguments

as_list	This options controls the output format. The output is either a named list or an atomic vector with the available sections
format	Specify a format type. The function now returns only available sections that can respond with that format. Currently geojson is the only format that is specified.
verbose	whether or not to provide {cli} output for the user. Enabled by default, but disabled for all functions that wrap this function
func	Sepcify the function you would like to ensure section compatability with.

Value

The function returns either a named list or an atomic vector with the available sections in the API. The format is decided by as_list.

Examples

```
available_sections()

available_sections(as_list = TRUE)
```

connection_check *Check that* **R** *has access to resources at* https://api.dataforsyningen.dk

Description

Check that **R** has access to resources at https://api.dataforsyningen.dk

Usage

```
connection_check()
```

Value

Returns a logical

See Also

Other Connection checks: `status_check()`

Examples

```
connection_check()
```

---

dawa *Base API function*

Description

This function is the foundation of all communication with the api in this package. Not intended for end-users, but it provides all the flexibility of the api without the nice conveniences of the other functions in this package. By default it reformats the api response (which is json) to a list object.

Usage

```
dawa(
  section,
  ...,
  append_to_url = NULL,
  format = NULL,
  verbose = TRUE,
  cache = TRUE,
  dry_run = FALSE,
  func_params = list()
)
```

Arguments

`section`	Defines what section of the api should be called. `available_sections()` exposes all options for this parameter.
`...`	<dynamic-dots> input query parameters.
`append_to_url`	This options enables you to append a string to the url. It is inserted in the request after the `section` but before the `....`

format	The format to provide as a query parameter for the api call. By default it has value NULL and triggers default json formatting. When geojson or geojsonz is provided, the function will return the raw json data instead of the usual list.
verbose	whether or not to provide {cli} output for the user. Enabled by default, but disabled for all functions that wrap this function
cache	Option to use caching in a tempdir(). Enabled by default. More info can be found in the documentation for httr2::req_cache().
dry_run	With this option enabled, the function will output the request that has been created - Without sending it to the api. This is useful for debugging.
func_params	Option to include extra parameters (just like ... above). This option is used to pass down the dot-inputs from get_data() and get_map_data(). It is recommended to use the dynamic dots instead of this option, as an end user.

Value

By default returns the API response as a list output. For geojson(z) formats it returns the raw json. When dry_run = TRUE it returns the request that has been created but does not run it.

Examples

```
if (connection_check()) {
  x <- dawa(section = "sogne")

  x[[1]]
}
```

---

get_data *Get data from section*

Description

Get data from section

Usage

```
get_data(section, as_list = FALSE, ...)
```

Arguments

section	Defines what section of the api should be called. available_sections() exposes all options for this parameter.
as_list	Whether or not to return the data as a list or in a dataframe. Data.frame is the default output.
...	<dynamic-dots> input query parameters.

Value

Returns either a dataframe with all the data for the specified section or a list with the same information.

See Also

Other Data functions: `get_map_data()`

Examples

```
if (connection_check()) {
  x <- get_data("regioner")
  head(x)
}
```

`get_map_data` *Fetch map data to use with* `{sf}`.

Description

This function returns map data from DAWA. It is ready to use with `{sf}` functions. The returned dataframe has date-time columns to inform when the information was last changed. It is returned in Europe/Copenhagen time as that is the *local time* for the API. You can change that behavior by setting `Sys.setenv(TZ = "Your/Timezone")`.

Usage

```
get_map_data(type, cache = TRUE, ...)
```

Arguments

`type`	Defines the type of map data to request from DAWA. Run `available_sections(format = "geojson")` to see your options.
`cache`	Boolean to determine whether or not to cache the api call and the function output. Default is `TRUE`.
`...`	<`dynamic-dots`> input query parameters.

Value

Returns a `data.frame` object that contains polygons (or points) for the section provided.

See Also

Other Data functions: `get_data()`

Examples

```
## Not run:
x <- get_map_data("regioner")
ggplot2::ggplot(x) +
  ggplot2::geom_sf()

## End(Not run)
```

`reverse` *Base function for interacting with* `reverse geocoding` *of sections.*

Description

This function reverse geocodes the coordinates provided. When used out of the box, the function wants WGS84 coordinates (lat/long). ETRS89 or UTM zone 32 can be provided as well. When coordinates from outside of Denmark is provided, the function will fail with a code `404`.

Usage

```
reverse(section, x, y, verbose = TRUE, type = NULL, ...)
```

Arguments

`section`	Defines what section of the api should be called. `available_sections()` exposes all options for this parameter.
`x`	The X coordinate to geocode. If no `srid` is provided this should be the lattitude.
`y`	The Y coordinate to geocode. If no `srid` is provided this should be the longitude.
`verbose`	whether or not to provide `{cli}` output for the user. Enabled by default, but disabled for all functions that wrap this function
`type`	The coordinate type for the API to parse. By default none is provided and `WGS84` is used. The options are `ETRS89`, `UTM32` and `WGS84`. The functions checks the input against the available types and errs if the type is not allowed.
`...`	<dynamic-dots> input query parameters.

Value

Returns a list object with information about the provided `section` for the location that is input.

Examples

```
if (connection_check()) {
  reverse("regioner", x = 12.58515, y = 55.68324)
}
```

`status_check` *Check that all DAWA services are operational*

Description

This function fetches the status of all available SDFI services. This package depends on at least one of these. It checks the official status page. The status can also be followed on their official X-page `@DriftDataforsyn`.

Usage

```
status_check(return_df = FALSE, error_if_unavailable = FALSE)
```

Arguments

`return_df`	This option makes the function return a data.frame object with the services and their statuses listed
`error_if_unavailable`	This option forces the function to throw an error in the event that one or more services are unreachable.

Value

Returns a data frame object with the services and their statuses. Is only returned if `return_df = TRUE`.

Note

Please note that the status at the status page is only checked at regular intervals. If you would like to know if your machine can access the serviceses, please try `connection_check()` as well.

See Also

Other Connection checks: `connection_check()`

Examples

```
if (connection_check()) {
  status_check()

  status_check(return_df = TRUE)
}
```

Index

∗ **Connection checks**
connection_check, 3
status_check, 8
∗ **Data functions**
get_data, 5
get_map_data, 6

autocomplete, 2
available_sections, 3

connection_check, 3
connection_check(), *8*

dawa, 4

get_data, 5
get_data(), *6*
get_map_data, 6
get_map_data(), *6*

reverse, 7

status_check, 8
status_check(), *4*