Package 'rnassqs'

May 9, 2026

Type Package

Title Access Data from the NASS 'Quick Stats' API

Version 0.6.3

Maintainer Nicholas Potter <econpotter@gmail.com>

Description Interface to access data via the United States Department of Agriculture's National Agricultural Statistical Service (NASS) 'Quick Stats' web API <https://quickstats.nass.usda.gov/api/>. Convenience functions facilitate building queries based on available parameters and valid parameter values. This product uses the NASS API but is not endorsed or certified by NASS.

URL https://docs.ropensci.org/rnassqs/ (website)
https://github.com/ropensci/rnassqs/

BugReports https://github.com/ropensci/rnassqs/issues

License MIT + file LICENSE

Language en-US

Depends R (>= 3.5.0)

Imports httr, jsonlite, stats, utils

Suggests here, httptest, knitr, rmarkdown, testthat

RoxygenNote 7.3.2

Encoding UTF-8

VignetteBuilder knitr

Collate 'auth.R' 'helpers.R' 'rnassqs-package.R' 'params.R' 'request.R' 'wrappers.R'

NeedsCompilation no

Author Nicholas Potter [aut, cre],
Robert Dinterman [ctb],
Jonathan Adams [ctb],
Joseph Stachelek [ctb],
Julia Piaskowski [ctb],
Branden Collingsworth [ctb],
Adam Sparks [rev],
Neal Richardson [ctb, rev]

Repository CRAN

Date/Publication 2024-08-30 03:50:02 UTC

Contents

`nassqs` *Get data and return a data frame*

Description

The primary function in the `rnassqs` package, `nassqs` makes a HTTP GET request to the USDA-NASS Quick Stats API and returns the data parsed as a data.frame, plain text, or list. Various other functions make use of `nassqs` to make specific queries. For a data request the Quick Stats API returns JSON that when parsed to a data.frame contains 39 columns and a varying number of rows depending on the query. Unfortunately there is not a way to restrict the number of columns.

Usage

```
nassqs(
  ...,
  agg_level_desc = NULL,
  asd_code = NULL,
  asd_desc = NULL,
  begin_code = NULL,
  class_desc = NULL,
  commodity_desc = NULL,
  congr_district_code = NULL,
  country_code = NULL,
  country_name = NULL,
  county_ansi = NULL,
  county_code = NULL,
  county_name = NULL,
```

```
  domaincat_desc = NULL,
  domain_desc = NULL,
  end_code = NULL,
  freq_desc = NULL,
  group_desc = NULL,
  load_time = NULL,
  location_desc = NULL,
  prodn_practice_desc = NULL,
  reference_period_desc = NULL,
  region_desc = NULL,
  sector_desc = NULL,
  short_desc = NULL,
  source_desc = NULL,
  state_alpha = NULL,
  state_ansi = NULL,
  state_fips_code = NULL,
  state_name = NULL,
  statisticcat_desc = NULL,
  unit_desc = NULL,
  util_practice_desc = NULL,
  watershed_code = NULL,
  watershed_desc = NULL,
  week_ending = NULL,
  year = NULL,
  zip_5 = NULL,
  as_numeric = TRUE,
  progress_bar = TRUE,
  format = "csv",
  as = "data.frame"
)
```

Arguments

`...`	either a named list of parameters or a series of additional parameters that include operations, e.g. `year__GE = 2010` for all records in 2010 and later. See `details` for information on available operators.
`agg_level_desc`	Geographic level ("AGRICULTURAL DISTRICT", "COUNTY", "INTERNATIONAL", "NATIONAL", "REGION : MULTI-STATE", "REGION : SUB-STATE", "STATE", "WATERSHED", or "ZIP CODE").
`asd_code`	Agriculture statistical district code.
`asd_desc`	Agriculture statistical district name / description.
`begin_code`	Week number indicating when the data series begins.
`class_desc`	Commodity class.
`commodity_desc`	Commodity, the primary subject of interest (e.g., "CORN", "CATTLE", "LABOR", "TRACTORS", "OPERATORS").
`congr_district_code`	Congressional District codes.

`country_code` Country code.

`country_name` Country name.

`county_ansi` County ANSI code.

`county_code` County FIPS code.

`county_name` County name.

`domaincat_desc` Domain category within a domain (e.g., under domain_desc = "SALES", domain categories include $1,000 TO $9,999, $10,000 TO $19,999, etc).

`domain_desc` Domain, a characteristic of operations that produce a particular commodity (e.g., "ECONOMIC CLASS", "AREA OPERATED", "NAICS CLASSIFICATION", "SALES"). For chemical usage data, the domain describes the type of chemical applied to the commodity. The domain_desc: = "TOTAL" will have no further breakouts; i.e., the data value pertains completely to the short_desc.

`end_code` = Week number that the data series ends.

`freq_desc` Time period type covered by the data ("ANNUAL", "SEASON", "MONTHLY", "WEEKLY", "POINT IN TIME"). "MONTHLY" often covers more than one month. "POINT IN TIME" is for a particular day.

`group_desc` Commodity group within a sector (e.g., under sector_desc = "CROPS", the groups are "FIELD CROPS", "FRUIT & TREE NUTS", "HORTICULTURE", and "VEGETABLES").

`load_time` Date and time of the data load, e.g. "2015-02-17 16:05:20".

`location_desc` Location code, e.g. 5-digit fips code for counties.

`prodn_practice_desc` Production practice, (e.g. "UNDER PROTECTION", "OWNED, RIGHTS, LEASED", "ORGANIC, TRANSITIONING", "HIRED MANAGER").

`reference_period_desc` Reference period of the data (e.g. "JUN", "MID SEP", "WEEK #32").

`region_desc` Region name (e.g. "TEXAS", "WA & OR", "WEST COAST", "UMATILLA").

`sector_desc` Sector, the five high level, broad categories useful to narrow down choices. ("ANIMALS & PRODUCTS", "CROPS", "DEMOGRAPHICS", "ECONOMICS", or "ENVIRONMENTAL").

`short_desc` A concatenation of six columns: `commodity_desc`, `class_desc`, `prodn_practice_desc`, `util_practice_desc`, `statisticcat_desc`, and `unit_desc`.

`source_desc` Source of data ("CENSUS" or "SURVEY"). Census program includes the Census of Ag as well as follow up projects. Survey program includes national, state, and county surveys.

`state_alpha` 2-character state abbreviation, e.g. "NM".

`state_ansi` State ANSI code.

`state_fips_code` State FIPS code.

`state_name` Full name of the state, e.g. "ALABAMA".

`statisticcat_desc` Statistical category of the data (e.g., "AREA HARVESTED", "PRICE RECEIVED", "INVENTORY", "SALES").

unit_desc	The units of the data (e.g. "TONS / ACRE", "TREES", "OPERATIONS", "NUMBER", "LB / ACRE", "BU / PLANTED ACRE").
util_practice_desc	Utilization practice (e.g. "WIND", "SUGAR", "SILAGE", "ONCE REFINED", "FEED", "ANIMAL FEED").
watershed_code	Watershed code as 8-digit HUC (e.g. "13020100").
watershed_desc	Watershed/HUC name (e.g. "UPPER COLORADO").
week_ending	Date of ending week (e.g. "1975-11-22").
year	Year of the data. Conditional values are possible by appending an operation to the parameter, e.g. "year__GE = 2020" will return all records with year >= 2020. See `details` for more on operations.
zip_5	5-digit zip code.
as_numeric	Whether to convert data to numeric format. Conversion will replace missing notation such as "(D)" or "(Z)" with NA, but removes the need to convert to numeric format after querying.
progress_bar	Whether or not to display the progress bar.
format	The format to return the query in. Only useful if as = "text".
as	whether to return a data.frame, list, or text string. See `nassqs_parse()`.

Details

`nassqs()` accepts all parameters that are accepted by the USDA-NASS Quick Stats. These parameters are listed in `nassqs_params()`, and are used to form the data query.

Parameters can be modified by operations, which are appended to the parameter name. For example, "year__GE = 2020" will fetch data in 2020 and after. Operations can take the following form:

- __LE: less than or equal (<=)
- __LT: less than (<)
- __GT: greater than (>)
- __GE: = >=
- __LIKE = like
- __NOT_LIKE = not like
- __NE = not equal

Value

a data frame, list, or text string of requested data.

See Also

`nassqs_GET()`, `nassqs_parse()`, `nassqs_yields()`, `nassqs_acres()`

Examples

```
## Not run:
  # Get corn yields in Virginia in 2012
  params <- list(commodity_desc = "CORN",
                 year = 2012,
                 agg_level_desc = "COUNTY",
                 state_alpha = "VA",
                 statisticcat_desc = "YIELD")
  yields <- nassqs(params)
  head(yields)

## End(Not run)
```

`nassqs_acres` *Get NASS Area given a set of parameters.*

Description

Get NASS Area given a set of parameters.

Usage

```
nassqs_acres(
  ...,
  area = c("AREA", "AREA PLANTED", "AREA BEARING", "AREA BEARING & NON-BEARING",
   "AREA GROWN", "AREA HARVESTED", "AREA IRRIGATED", "AREA NON-BEARING", "AREA PLANTED",
    "AREA PLANTED, NET")
)
```

Arguments

`...`	either a named list of parameters or a series of parameters to form the query
`area`	the type of area to return. Default is all types.

Value

a data.frame of acres data

Examples

```
## Not run:
  # Get Area bearing for Apples in Washington, 2012.
  params <- list(
    commodity_desc = "APPLES",
    year = "2012",
    state_name = "WASHINGTON",
    agg_level_desc = "STATE"
  )
```

```
  area <- nassqs_acres(params, area = "AREA BEARING")
  head(area)

## End(Not run)
```

`nassqs_auth` *Get/Set the environmental variable NASSQS_TOKEN to the API key*

Description

If the API key is provided, sets the environmental variable. You can set your API key in four ways:

Usage

```
nassqs_auth(key)
```

Arguments

`key`	the API key (obtained from https://quickstats.nass.usda.gov/api/)

Details

1. directly or as a variable from your R program: `nassqs_auth(key = "<your api key>"`
2. by setting `NASSQS_TOKEN` in your R environment file (you'll never have to enter it again).
3. by entering it into the console when asked (it will be stored for the rest of the session.)

Examples

```
# Set the API key
nassqs_auth(key = "<your api key>")
Sys.getenv("NASSQS_TOKEN")
```

`nassqs_byfips` *Allow querying for a given set of counties based on FIPS.*

Description

This wrapper allows specifying a list of counties by FIPS code. It iterates over each state in the list of FIPS, downloading for each separately and then concatenating.

Usage

```
nassqs_byfips(fips, ...)
```

Arguments

fips	a list of 5-digit fips codes
...	either a named list of parameters or a series of parameters to form the query

Value

a data.frame of data for each fips code

Examples

```
## Not run:
nassqs_byfips(
  fips = c("19001", "17005", "17001"),
  commodity_desc = "CORN",
  year = 2019,
  statisticcat_desc = "YIELD")

## End(Not run)
```

---

nassqs_check *Check the response.*

Description

Check that the response is valid, i.e. that it doesn't exceed 50,000 records and that all the parameter values are valid. This is used to ensure that the query is valid before querying to reduce wait times before receiving an error.

Usage

```
nassqs_check(response)
```

Arguments

response	a `httr::GET()` request result returned from the API.

Value

nothing if check is passed, or an informative error if not passed.

nassqs_fields *Deprecated: Return list of NASS QS parameters.*

Description

Deprecated. Use `nassqs_params()` instead.

Usage

```
nassqs_fields(...)
```

Arguments

`...`	a parameter, series of parameters, or a list of parameters that you would like a description of. If missing, a list of all available parameters is returned.

nassqs_GET *Issue a GET request to the NASS 'Quick Stats' API*

Description

This is the workhorse of the package that provides the core request functionality to the NASS 'Quick Stats' API: https://quickstats.nass.usda.gov/api/. In most cases `nassqs()` or other high-level functions should be used. `nassqs_GET()` uses `httr::GET()` to make a HTTP GET request, which returns a request object which must then be parsed to a data.frame, list, or other `R` object. Higher-level functions will do that parsing automatically. However, if you need access to the request object directly, `nassqs_GET()` provides that.

Usage

```
nassqs_GET(
  ...,
  api_path = c("api_GET", "get_param_values", "get_counts"),
  progress_bar = TRUE,
  format = c("csv", "json", "xml")
)
```

Arguments

`...`	either a named list of parameters or a series of parameters to use in the query
`api_path`	the API path that determines the type of request being made.
`progress_bar`	whether to display the project bar or not.
`format`	The format to return the query in. Only useful if as = "text".

Value

a `httr::GET()` response object

Examples

```
## Not run:
  # Yields for corn in 2012 in Washington
  params <- list(commodity_desc = "CORN",
                 year = 2012,
                 agg_level_desc = "STATE",
                 state_alpha = "WA",
                 statisticcat_desc = "YIELD")

  # Returns a request object that must be parsed either manually or
  # by using nassqs_parse()
  response <- nassqs_GET(params)
  yields <- nassqs_parse(response)
  head(yields)

  # Get the number of records that would be returned for a given request
  # Equivalent to 'nassqs_record_count(params)'
  response <- nassqs_GET(params, api_path = "get_counts")
  records <- nassqs_parse(response)
  records

  # Get the list of allowable values for the parameters 'statisticcat_desc'
  # Equivalent to 'nassqs_param_values("statisticcat_desc")'
  req <- nassqs_GET(list(param = "statisticcat_desc"),
                    api_path = "get_param_values")
  statisticcat_desc_values <- nassqs_parse(req, as = "list")
  head(statisticcat_desc_values)

## End(Not run)
```

`nassqs_params` *Return list of NASS QS parameters.*

Description

Contains a simple hard-coded list of all available parameters. If no parameter name is provided, returns a list of all parameters. More information can be found in the API documentation on parameters found at https://quickstats.nass.usda.gov/api/#param_define.

Usage

```
nassqs_params(...)
```

Arguments

...	a parameter, series of parameters, or a list of parameters that you would like a description of. If missing, a list of all available parameters is returned.

Value

a list of all available parameters or a description of a subset

Examples

```
# Get a list of all available parameters
nassqs_params()

# Get information about specific parameters
nassqs_params("source_desc", "group_desc")
```

---

nassqs_param_values *Get all values for a specific parameter.*

Description

Returns a list of all possible values for a given parameter. Including additional parameters will restrict the list of valid values to those for data meeting the additional parameter restrictions. However, this is only possible by requesting the entire dataset and then filtering for unique values. It is recommended to make the query as small as possible if including additional parameters

Usage

```
nassqs_param_values(param, ...)
```

Arguments

param	the name of a NASS quickstats parameter
...	additional parameters for which to filter the valid responses.

Value

a list containing all valid values for that parameter

Examples

```
## Not run:
  # See all values available for the statisticcat_desc field. Values may not
  # be available in the context of other parameters you set, for example
  # a given state may not have any 'YIELD' in blueberries if they don't grow
  # blueberries in that state.
  # Requires an API key:
```

```
  nassqs_param_values("source_desc")

  # Valid values for a parameter given a specific set of additional
  # parameters
  nassqs_param_values("commodity_desc", state_fips_code = "53",
                      county_code = "077", year = 2017,
                      group_desc = "EXPENSES")

## End(Not run)
```

`nassqs_parse` *Parse a response object from* `nassqs_GET()`.

Description

Returns a data frame, list, or text string. If a data.frame, all columns except `year` strings because the 'Quick Stats' data returns suppressed data as '(D)', '(Z)', or other character indicators which mean different things. Converting the value to a numerical results in NA, which loses that information.

Usage

```
nassqs_parse(req, as_numeric = TRUE, as = c("data.frame", "list", "text"), ...)
```

Arguments

`req`	the GET response from `nassqs_GET()`
`as_numeric`	whether to convert values to numeric format.
`as`	whether to return a data.frame, list, or text string
`...`	additional parameters passed to `jsonlite::fromJSON()` or `utils::read.csv()`

Value

a data frame, list, or text string of the content from the response.

Examples

```
## Not run:
  # Set parameters and make the request
  params <- list(commodity_desc = "CORN",
                 year = 2012,
                 agg_level_desc = "STATE",
                 state_alpha = "WA",
                 statisticcat_desc = "YIELD")
  response <- nassqs_GET(params)

  # Parse the response to a data frame
  corn <- nassqs_parse(response, as = "data.frame")
  head(corn)
```

```
  # Parse the response into a raw character string.
  corn_text<- nassqs_parse(response, as = "text")
  head(corn_text)

  # Get a list of parameter values and parse as a list
  response <- nassqs_GET(list(param = "statisticcat_desc"),
                   api_path = "get_param_values")
  statisticcat_desc_values <- nassqs_parse(response, as = "list")
  head(statisticcat_desc_values)

## End(Not run)
```

`nassqs_record_count` *Get a count of number of records for given parameters.*

Description

Returns the number of records that fit a set of parameters. Useful if your current parameter set returns more than the 50,000 record limit.

Usage

```
nassqs_record_count(...)
```

Arguments

`...`	either a named list of parameters or a series of parameters to form the query

Value

integer that is the number of records that are returned from the API in response to the query

Examples

```
## Not run:
  # Check the number of records returned for corn in 1995, Washington state
  params <- list(
    commodity_desc = "CORN",
    year = "2005",
    agg_level_desc = "STATE",
    state_name = "WASHINGTON"
  )

  records <- nassqs_record_count(params)
  records  # returns 17

## End(Not run)
```

nassqs_yields *Get yield records for a specified crop.*

Description

Returns yields for other specified parameters. This function is intended to simplify common requests.

Usage

```
nassqs_yields(...)
```

Arguments

...	either a named list of parameters or a series of parameters to form the query

Value

a data.frame of yields data

Examples

```
## Not run:
  # Get yields for wheat in 2012, all geographies
  params <- list(
    commodity_desc = "WHEAT",
    year = "2012",
    agg_level_desc = "STATE",
    state_alpha = "WA")

  yields <- nassqs_yields(params)
  head(yields)

## End(Not run)
```

Index