Package 'googlesheets4'

March 21, 2020

Title Access Google Sheets using the Sheets API V4

Version 0.1.1

Description Interact with Google Sheets through the Sheets API v4 <https://developers.google.com/sheets/api>. ``API'' is an acronym for ``application programming interface''; the Sheets API allows users to interact with Google Sheets programmatically, instead of via a web browser. The ``v4'' refers to the fact that the Sheets API is currently at version 4. This package helps the user to retrieve Sheet metadata and to read data out of specific worksheets or ranges into an R object, such as a data frame.

License MIT + file LICENSE

URL https://github.com/tidyverse/googlesheets4

BugReports https://github.com/tidyverse/googlesheets4/issues

Depends R (>= 3.2)

Imports cellranger, curl, gargle (>= 0.4.0), glue (>= 1.3.0), googledrive (>= 1.0.0), httr, magrittr, purrr, rematch2, rlang, tibble (>= 2.1.1), utils

Suggests covr, rmarkdown, sodium, spelling, testthat (>= 2.1.0)

ByteCompile true

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.1.0

NeedsCompilation no

Author Jennifer Bryan [cre, aut] (<https://orcid.org/0000-0002-6983-2759>), RStudio [cph, fnd]

Maintainer Jennifer Bryan <jenny@rstudio.com>

Repository CRAN

Date/Publication 2020-03-21 05:40:02 UTC

R topics documented:

`as_sheets_id` *Coerce to a sheets_id object*

Description

Converts various representations of a Google Sheet into a `sheets_id` object. Anticipated inputs:

- Spreadsheet id, "a string containing letters, numbers, and some special characters", typically 44 characters long, in our experience. Example: 1qpyC0XzvTcKT6EISywvqESX3A0MwQoFDE8p-Bll4hps.
- A URL, from which we can excavate a spreadsheet or file id. Example: `https://docs.google.com/spreadsheets/d/1BzfL0kZUz1TsI5zxJF1WNF01IxvC67FbOJUiiGMZ_mQ/edit#gid=1150108545`.
- A one-row `dribble`, a "Drive tibble" used by the googledrive package. In general, a `dribble` can represent several files, one row per file. Since googlesheets4 is not vectorized over spreadsheets, we are only prepared to accept a one-row `dribble`.
  - `googledrive::drive_get("YOUR SHEET NAME")` is a great way to look up a Sheet via its name.

This is a generic function.

Usage

```
as_sheets_id(x, ...)
```

Arguments

x	Something that uniquely identifies a Google Sheet: a sheets_id, URL, or dribble.
...	Other arguments passed down to methods. (Not used.)

Examples

```
as_sheets_id("abc")
```

---

cell-specification *Specify cells for reading*

Description

The range argument in read_sheet() or sheets_cells() is used to limit the read to a specific rectangle of cells. The Sheets v4 API only accepts ranges in A1 notation, but googlesheets4 accepts and converts a few alternative specifications provided by the functions in the cellranger package. Of course, you can always provide A1-style ranges directly to functions like read_sheet() or sheets_cells(). Why would you use the cellranger helpers? Some ranges are practically impossible to express in A1 notation, specifically when you want to describe rectangles with some bounds that are specified and others determined by the data.

Examples

```
if (sheets_has_token() && interactive()) {
  ss <- sheets_example("mini-gap")

  # Specify only the rows or only the columns
  read_sheet(ss, range = cell_rows(1:3))
  read_sheet(ss, range = cell_cols("C:D"))
  read_sheet(ss, range = cell_cols(1))

  # Specify upper or lower bound on row or column
  read_sheet(ss, range = cell_rows(c(NA, 4)))
  read_sheet(ss, range = cell_cols(c(NA, "D")))
  read_sheet(ss, range = cell_rows(c(3, NA)))
  read_sheet(ss, range = cell_cols(c(2, NA)))
  read_sheet(ss, range = cell_cols(c("C", NA)))

  # Specify a partially open rectangle
  read_sheet(ss, range = cell_limits(c(2, 3), c(NA, NA)), col_names = FALSE)
  read_sheet(ss, range = cell_limits(c(1, 2), c(NA, 4)))
}
```

read_sheet *Read a Sheet into a data frame*

Description

This is the main "read" function of the googlesheets4 package. The goal is that `read_sheet()` is to a Google Sheet as `readr::read_csv()` is to a csv file or `readxl::read_excel()` is to an Excel spreadsheet.

Usage

```
read_sheet(
  ss,
  sheet = NULL,
  range = NULL,
  col_names = TRUE,
  col_types = NULL,
  na = "",
  trim_ws = TRUE,
  skip = 0,
  n_max = Inf,
  guess_max = min(1000, n_max),
  .name_repair = "unique"
)

sheets_read(
  ss,
  sheet = NULL,
  range = NULL,
  col_names = TRUE,
  col_types = NULL,
  na = "",
  trim_ws = TRUE,
  skip = 0,
  n_max = Inf,
  guess_max = min(1000, n_max),
  .name_repair = "unique"
)
```

Arguments

ss	Something that identifies a Google Sheet: its file ID, a URL from which we can recover the ID, or a dribble, which is how googledrive represents Drive files. Processed through as_sheets_id().
sheet	Sheet to read, as in "worksheet" or "tab". Either a string (the name of a sheet), or an integer (the position of the sheet). Ignored if the sheet is specified via range. If neither argument specifies the sheet, defaults to the first visible sheet.

range	A cell range to read from. If NULL, all non-empty cells are read. Otherwise specify range as described in Sheets A1 notation or using the helpers documented in cell-specification. Sheets uses fairly standard spreadsheet range notation, although a bit different from Excel. Examples of valid ranges: "Sheet1!A1:B2", "Sheet1!A:A", "Sheet1!1:2", "Sheet1!A5:A", "A1:B2", "Sheet1". Interpreted strictly, even if the range forces the inclusion of leading, trailing, or embedded empty rows or columns. Takes precedence over skip, n_max and sheet. Note range can be a named range, like "sales_data", without any cell reference.
col_names	TRUE to use the first row as column names, FALSE to get default names, or a character vector to provide column names directly. If user provides col_types, col_names can have one entry per column or one entry per unskipped column.
col_types	Column types. Either NULL to guess all from the spreadsheet or a string of readr-style shortcodes, with one character or code per column. If exactly one col_type is specified, it is recycled. See Details for more.
na	Character vector of strings to interpret as missing values. By default, blank cells are treated as missing data.
trim_ws	Logical. Should leading and trailing whitespace be trimmed from cell contents?
skip	Minimum number of rows to skip before reading anything, be it column names or data. Leading empty rows are automatically skipped, so this is a lower bound. Ignored if range is given.
n_max	Maximum number of data rows to parse into the returned tibble. Trailing empty rows are automatically skipped, so this is an upper bound on the number of rows in the result. Ignored if range is given. n_max is imposed locally, after reading all non-empty cells, so, if speed is an issue, it is better to use range.
guess_max	Maximum number of data rows to use for guessing column types.
.name_repair	Handling of column names. By default, googlesheets4 ensures column names are not empty and are unique. There is full support for .name_repair as documented in tibble::tibble().

Value

A tibble

Column specification

Column types must be specified in a single string of readr-style short codes, e.g. "cci?l" means "character, character, integer, guess, logical". This is not where googlesheets4's col spec will end up, but it gets the ball rolling in a way that is consistent with readr and doesn't reinvent any wheels.

Shortcodes for column types:

- _ or -: Skip. Data in a skipped column is still requested from the API (the high-level functions in this package are rectangle-oriented), but is not parsed into the data frame output.
- ?: Guess. A type is guessed for each cell and then a consensus type is selected for the column. If no atomic type is suitable for all cells, a list-column is created, in which each cell is converted to an R object of "best" type". If no column types are specified, i.e. col_types = NULL, all types are guessed.

- `l`: Logical.
- `i`: Integer. This type is never guessed from the data, because Sheets have no formal cell type for integers.
- `d` or `n`: Numeric, in the sense of "double".
- `D`: Date. This type is never guessed from the data, because date cells are just serial datetimes that bear a "date" format.
- `t`: Time of day. This type is never guessed from the data, because time cells are just serial datetimes that bear a "time" format. *Not implemented yet; returns POSIXct.*
- `T`: Datetime, specifically POSIXct.
- `c`: Character.
- `C`: Cell. This type is unique to googlesheets4. This returns raw cell data, as an R list, which consists of everything sent by the Sheets API for that cell. Has S3 type of `"CELL_SOMETHING"` and `"SHEETS_CELL"`. Mostly useful internally, but exposed for those who want direct access to, e.g., formulas and formats.
- `L`: List, as in "list-column". Each cell is a length-1 atomic vector of its discovered type.
- *Still to come*: duration (code will be `:`) and factor (code will be `f`).

Examples

```
if (sheets_has_token()) {
  ss <- sheets_example("deaths")
  read_sheet(ss, range = "A5:F15")
  read_sheet(ss, range = "other!A5:F15", col_types = "ccilDD")
  read_sheet(ss, range = "arts_data", col_types = "ccilDD")

  read_sheet(sheets_example("mini-gap"))
  read_sheet(
    sheets_example("mini-gap"),
    sheet = "Europe",
    range = "A:D",
    col_types = "ccid"
  )
}
```

`request_generate` *Generate a Google Sheets API request*

Description

Generate a request, using knowledge of the Sheets API from its Discovery Document. Use `request_make()` to execute the request. Most users should, instead, use higher-level wrappers that facilitate common tasks, such as reading or writing worksheets or cell ranges. The functions here are intended for internal use and for programming around the Sheets API.

`request_generate()` lets you provide the bare minimum of input. It takes a nickname for an endpoint and:

- Uses the API spec to look up the `method`, `path`, and `base_url`.
- Checks `params` for validity and completeness with respect to the endpoint. Uses `params` for URL endpoint substitution and separates remaining parameters into those destined for the body versus the query.
- Adds an API key to the query if and only if `token = NULL`.

Usage

```
request_generate(
  endpoint = character(),
  params = list(),
  key = NULL,
  token = sheets_token()
)
```

Arguments

`endpoint`	Character. Nickname for one of the selected Sheets API v4 endpoints built into googlesheets4. Learn more in `sheets_endpoints()`.
`params`	Named list. Parameters destined for endpoint URL substitution, the query, or the body.
`key`	API key. Needed for requests that don't contain a token. The need for an API key in the absence of a token is explained in Google's document Credentials, access, security, and identity. In order of precedence, these sources are consulted: the formal `key` argument, a `key` parameter in `params`, a user-configured API key set up with `sheets_auth_configure()` and retrieved with `sheets_api_key()`.
`token`	Set this to `NULL` to suppress the inclusion of a token. Note that, if auth has been de-activated via `sheets_deauth()`, `sheets_token()` will actually return `NULL`.

Value

`list()`
Components are `method`, `url`, `body`, and `token`, suitable as input for `request_make()`.

See Also

`gargle::request_develop()`, `gargle::request_build()`, `gargle::request_make()`

Other low-level API functions: `request_make()`, `sheets_has_token()`, `sheets_token()`

Examples

```
req <- request_generate(
  "sheets.spreadsheets.get",
  list(spreadsheetId = sheets_example("deaths")),
  token = NULL
)
req
```

request_make *Make a Google Sheets API request*

Description

Low-level function to execute a Sheets API request. Most users should, instead, use higher-level wrappers that facilitate common tasks, such as reading or writing worksheets or cell ranges. The functions here are intended for internal use and for programming around the Sheets API.

`make_request()` does very, very little: it calls an HTTP method, only adding the googlesheets4 user agent. Typically the input has been created with `request_generate()` or `gargle::request_build()` and the output is processed with `process_response()`.

Usage

```
request_make(x, ...)
```

Arguments

x	List. Holds the components for an HTTP request, presumably created with `request_generate()` or `gargle::request_build()`. Must contain a `method` and `url`. If present, `body` and `token` are used.
...	Optional arguments passed through to the HTTP method.

Value

Object of class `response` from httr.

See Also

Other low-level API functions: `request_generate()`, `sheets_has_token()`, `sheets_token()`

sheets_auth *Authorize googlesheets4*

Description

Authorize googlesheets4 to view and manage your Google Sheets. This function is a wrapper around `gargle::token_fetch()`.

By default, you are directed to a web browser, asked to sign in to your Google account, and to grant googlesheets4 permission to operate on your behalf with Google Sheets. By default, these user credentials are cached in a folder below your home directory, ~/.R/gargle/gargle-oauth, from where they can be automatically refreshed, as necessary. Storage at the user level means the same token can be used across multiple projects and tokens are less likely to be synced to the cloud by accident.

If you are interacting with R from a web-based platform, like RStudio Server or Cloud, you need to use a variant of this flow, known as out-of-band auth ("oob"). If this does not happen automatically, you can request it yourself with `use_oob = TRUE` or, more persistently, by setting an option via `options(gargle_oob_default = TRUE)`.

Usage

```
sheets_auth(
  email = gargle::gargle_oauth_email(),
  path = NULL,
  scopes = "https://www.googleapis.com/auth/spreadsheets",
  cache = gargle::gargle_oauth_cache(),
  use_oob = gargle::gargle_oob_default(),
  token = NULL
)
```

Arguments

email	Optional. Allows user to target a specific Google identity. If specified, this is used for token lookup, i.e. to determine if a suitable token is already available in the cache. If no such token is found, email is used to pre-select the targetted Google identity in the OAuth chooser. Note, however, that the email associated with a token when it's cached is always determined from the token itself, never from this argument. Use NA or FALSE to match nothing and force the OAuth dance in the browser. Use TRUE to allow email auto-discovery, if exactly one matching token is found in the cache. Defaults to the option named "gargle_oauth_email", retrieved by gargle::gargle_oauth_email().
path	JSON identifying the service account, in one of the forms supported for the txt argument of jsonlite::fromJSON() (typically, a file path or JSON string).
scopes	A character vector of scopes to request. Pick from those listed at https://developers.google.com/identity/protocols/googlescopes. For certain token flows, the "https://www.googleapis.com/auth/userinfo.email" scope is unconditionally included. This grants permission to retrieve the email address associated with a token; gargle uses this to index cached OAuth tokens. This grants no permission to view or send email. It is considered a low value scope and does not appear on the consent screen.
cache	Specifies the OAuth token cache. Defaults to the option named "gargle_oauth_cache", retrieved via gargle::gargle_oauth_cache().
use_oob	Whether to prefer "out of band" authentication. Defaults to the option named "gargle_oob_default", retrieved via gargle::gargle_oob_default().
token	A token with class Token2.0 or an object of httr's class request, i.e. a token that has been prepared with httr::config() and has a Token2.0 in the auth_token component.

Details

Most users, most of the time, do not need to call sheets_auth() explicitly – it is triggered by the first action that requires authorization. Even when called, the default arguments often suffice. However, when necessary, this function allows the user to explicitly:

- Declare which Google identity to use, via an email address. If there are multiple cached tokens, this can clarify which one to use. It can also force googlesheets4 to switch from one identity to another. If there's no cached token for the email, this triggers a return to the browser to choose the identity and give consent.

- Use a service account token.
- Bring their own Token2.0.
- Specify non-default behavior re: token caching and out-of-bound authentication.

For details on the many ways to find a token, see `gargle::token_fetch()`. For deeper control over auth, use `sheets_auth_configure()` to bring your own OAuth app or API key. Read more about gargle options, see gargle::gargle_options.

See Also

Other auth functions: `sheets_auth_configure()`, `sheets_deauth()`

Examples

```
if (interactive()) {
  # load/refresh existing credentials, if available
  # otherwise, go to browser for authentication and authorization
  sheets_auth()

  # force use of a token associated with a specific email
  sheets_auth(email = "jenny@example.com")

  # use a 'read only' scope, so it's impossible to edit or delete Sheets
  sheets_auth(
    scopes = "https://www.googleapis.com/auth/spreadsheets.readonly"
  )

  # use a service account token
  sheets_auth(path = "foofy-83ee9e7c9c48.json")
}
```

`sheets_auth_configure` *Edit and view auth configuration*

Description

These functions give more control over and visibility into the auth configuration than `sheets_auth()` does. `sheets_auth_configure()` lets the user specify their own:

- OAuth app, which is used when obtaining a user token.
- API key. If googlesheets4 is de-authorized via `sheets_deauth()`, all requests are sent with an API key in lieu of a token. See the vignette How to get your own API credentials for more. If the user does not configure these settings, internal defaults are used. `sheets_oauth_app()` and `sheets_api_key()` retrieve the currently configured OAuth app and API key, respectively.

Usage

```
sheets_auth_configure(app, path, api_key)

sheets_api_key()

sheets_oauth_app()
```

Arguments

app	OAuth app, in the sense of `httr::oauth_app()`.
path	JSON downloaded from Google Cloud Platform Console, containing a client id (aka key) and secret, in one of the forms supported for the `txt` argument of `jsonlite::fromJSON()` (typically, a file path or JSON string).
api_key	API key.

Value

- `sheets_auth_configure()`: An object of R6 class gargle::AuthState, invisibly.
- `sheets_oauth_app()`: the current user-configured `httr::oauth_app()`.
- `sheets_api_key()`: the current user-configured API key.

See Also

Other auth functions: `sheets_auth()`, `sheets_deauth()`

Examples

```
# see and store the current user-configured OAuth app (probaby `NULL`)
(original_app <- sheets_oauth_app())

# see and store the current user-configured API key (probaby `NULL`)
(original_api_key <- sheets_api_key())

if (require(httr)) {
  # bring your own app via client id (aka key) and secret
  google_app <- httr::oauth_app(
    "my-awesome-google-api-wrapping-package",
    key = "YOUR_CLIENT_ID_GOES_HERE",
    secret = "YOUR_SECRET_GOES_HERE"
  )
  google_key <- "YOUR_API_KEY"
  sheets_auth_configure(app = google_app, api_key = google_key)

  # confirm the changes
  sheets_oauth_app()
  sheets_api_key()

  # bring your own app via JSON downloaded from Google Developers Console
  # this file has the same structure as the JSON from Google
```

```
  app_path <- system.file(
    "extdata", "fake-oauth-client-id-and-secret.json",
    package = "googlesheets4"
  )
  sheets_auth_configure(path =app_path)

  # confirm the changes
  sheets_oauth_app()
}

# restore original auth config
sheets_auth_configure(app = original_app, api_key = original_api_key)
```

`sheets_browse` *Visit Sheet in browser*

Description

Visits a Google Sheet in your default browser, if session is interactive.

Usage

```
sheets_browse(ss)
```

Arguments

`ss`	Something that identifies a Google Sheet: its file ID, a URL from which we can recover the ID, or a `dribble`, which is how googledrive represents Drive files. Processed through `as_sheets_id()`.

Value

The Sheet's browser URL, invisibly.

Examples

```
sheets_example("mini-gap") %>% sheets_browse()
```

sheets_cells *Read cells from a Sheet*

Description

This low-level function returns cell data in a tibble with integer variables `row` and `column` (referring to location with the Google Sheet), an A1-style reference `loc`, and a `cell` list-column. The flagship function `read_sheet()`, a.k.a. `sheets_read()`, is what most users are looking for. It is basically `sheets_cells()` (this function), followed by `spread_sheet()`, which looks after reshaping and column typing. But if you want the raw data from the API, use `sheets_cells()`.

Usage

```
sheets_cells(ss, sheet = NULL, range = NULL, skip = 0, n_max = Inf)
```

Arguments

`ss`	Something that identifies a Google Sheet: its file ID, a URL from which we can recover the ID, or a `dribble`, which is how googledrive represents Drive files. Processed through `as_sheets_id()`.
`sheet`	Sheet to read, as in "worksheet" or "tab". Either a string (the name of a sheet), or an integer (the position of the sheet). Ignored if the sheet is specified via `range`. If neither argument specifies the sheet, defaults to the first visible sheet.
`range`	A cell range to read from. If `NULL`, all non-empty cells are read. Otherwise specify `range` as described in Sheets A1 notation or using the helpers documented in cell-specification. Sheets uses fairly standard spreadsheet range notation, although a bit different from Excel. Examples of valid ranges: `"Sheet1!A1:B2"`, `"Sheet1!A:A"`, `"Sheet1!1:2"`, `"Sheet1!A5:A"`, `"A1:B2"`, `"Sheet1"`. Interpreted strictly, even if the range forces the inclusion of leading, trailing, or embedded empty rows or columns. Takes precedence over `skip`, `n_max` and `sheet`. Note `range` can be a named range, like `"sales_data"`, without any cell reference.
`skip`	Minimum number of rows to skip before reading anything, be it column names or data. Leading empty rows are automatically skipped, so this is a lower bound. Ignored if `range` is given.
`n_max`	Maximum number of data rows to parse into the returned tibble. Trailing empty rows are automatically skipped, so this is an upper bound on the number of rows in the result. Ignored if `range` is given. `n_max` is imposed locally, after reading all non-empty cells, so, if speed is an issue, it is better to use `range`.

Value

A tibble with one row per non-empty cell in the `range`.

Examples

```
if (sheets_has_token()) {
  sheets_cells(sheets_example("deaths"), range = "arts_data")

  sheets_example("cell-contents-and-formats") %>%
    sheets_cells(range = "types!A2:A5")
}
```

`sheets_deauth` *Suspend authorization*

Description

Put googlesheets4 into a de-authorized state. Instead of sending a token, googlesheets4 will send an API key. This can be used to access public resources for which no Google sign-in is required. This is handy for using googlesheets4 in a non-interactive setting to make requests that do not require a token. It will prevent the attempt to obtain a token interactively in the browser. The user can configure their own API key via `sheets_auth_configure()` and retrieve that key via `sheets_api_key()`. In the absence of a user-configured key, a built-in default key is used.

Usage

```
sheets_deauth()
```

See Also

Other auth functions: `sheets_auth_configure()`, `sheets_auth()`

Examples

```
if (interactive()) {
  sheets_deauth()
  sheets_user()

  # get metadata on the public 'deaths' spreadsheet
  sheets_get(sheets_example("deaths"))
}
```

sheets_endpoints *List Sheets endpoints*

Description

Returns a list of selected Sheets API v4 endpoints, as stored inside the googlesheets4 package. The names of this list (or the `id` sub-elements) are the nicknames that can be used to specify an endpoint in `request_generate()`. For each endpoint, we store its nickname or `id`, the associated HTTP `method`, the `path`, and details about the parameters. This list is derived programmatically from the Sheets API v4 Discovery Document.

Usage

```
sheets_endpoints(i = NULL)
```

Arguments

`i`	The name(s) or integer index(ices) of the endpoints to return. Optional. By default, the entire list is returned.

Value

A list containing some or all of the subset of the Sheets API v4 endpoints that are used internally by googlesheets4.

Examples

```
str(sheets_endpoints(), max.level = 2)
sheets_endpoints("sheets.spreadsheets.values.get")
sheets_endpoints(4)
```

sheets_example *File IDs of example Sheets*

Description

googlesheets4 ships with static IDs for some world-readable example Sheets for use in examples and documentation. These functions make them easy to access by a nickname.

Usage

```
sheets_example(name = names(sheets_examples()))

sheets_examples()
```

Arguments

name	Nickname of an example Sheet.

Value

- sheets_example(): a single sheets_id object
- sheets_examples(): a named character vector of all built-in examples

Examples

```
sheets_examples()

sheets_example("gapminder")
```

---

sheets_find *Find Google Sheets*

Description

Finds your Google Sheets. This is a very thin wrapper around googledrive::drive_find(), that specifies you want to list Drive files where type = "spreadsheet". Therefore, note that this will require auth for googledrive! See the article Using googlesheets4 with googledrive if you want to coordinate auth between googlesheets4 and googledrive.

Usage

```
sheets_find(...)
```

Arguments

...	Arguments (other than type, which is hard-wired as type = "spreadsheet") that are passed along to googledrive::drive_find().

Value

An object of class dribble, a tibble with one row per item.

Examples

```
if (sheets_has_token()) {
  # see all your Sheets
  sheets_find()

  # see 5 Sheets, prioritized by creation time
  x <- sheets_find(order_by = "createdTime desc", n_max = 5)
  x

  # hoist the creation date, using other packages in the tidyverse
```

```
  # x %>%
  #   tidyr::hoist(drive_resource, created_on = "createdTime") %>%
  #   dplyr::mutate(created_on = as.Date(created_on))
}
```

`sheets_get` *Get Sheet metadata*

Description

Retrieve spreadsheet-specific metadata, such as details on the individual (work)sheets or named ranges.

- `sheets_get()` complements `googledrive::drive_get()`, which returns metadata that exists for any file on Drive.
- `sheets_sheets()` is a very focused function that only returns (work)sheet names.

Usage

```
sheets_get(ss)

sheets_sheets(ss)
```

Arguments

`ss`	Something that identifies a Google Sheet: its file ID, a URL from which we can recover the ID, or a `dribble`, which is how googledrive represents Drive files. Processed through `as_sheets_id()`.

Value

- `sheets_get()`: A list with S3 class `sheets_meta`, for printing purposes.
- `sheets_sheets()`: A character vector.

Examples

```
if (sheets_has_token()) {
  sheets_get(sheets_example("mini-gap"))
}
if (sheets_has_token()) {
  sheets_sheets(sheets_example("deaths"))
}
```

sheets_has_token *Is there a token on hand?*

Description

Reports whether googlesheets4 has stored a token, ready for use in downstream requests.

Usage

```
sheets_has_token()
```

Value

Logical.

See Also

Other low-level API functions: `request_generate()`, `request_make()`, `sheets_token()`

Examples

```
sheets_has_token()
```

sheets_id `sheets_id` *object*

Description

Holds a spreadsheet identifier, i.e. a string. This is what the Sheets and Drive APIs refer to as `spreadsheetId` and `fileId`, respectively. Any object of class `sheets_id` will also have the `drive_id` class, which is used by googledrive for the same purpose.

This means you can pipe a `sheets_id` object straight into googledrive functions for all your Google Drive needs that have nothing to do with the file being a spreadsheet. Examples: examine or change file name, path, or permissions, copy the file, or visit it in a web browser.

See Also

`as_sheets_id()`

Examples

```
sheets_example("mini-gap")
```

sheets_token *Produce configured token*

Description

For internal use or for those programming around the Sheets API. Returns a token pre-processed with `httr::config()`. Most users do not need to handle tokens "by hand" or, even if they need some control, `sheets_auth()` is what they need. If there is no current token, `sheets_auth()` is called to either load from cache or initiate OAuth2.0 flow. If auth has been deactivated via `sheets_deauth()`, `sheets_token()` returns `NULL`.

Usage

```
sheets_token()
```

Value

A `request` object (an S3 class provided by httr).

See Also

Other low-level API functions: `request_generate()`, `request_make()`, `sheets_has_token()`

Examples

```
if (sheets_has_token()) {
  req <- request_generate(
    "sheets.spreadsheets.get",
    list(spreadsheetId = "abc"),
    token = sheets_token()
  )
  req
}
```

sheets_user *Get info on current user*

Description

Reveals the email address of the user associated with the current token. If no token has been loaded yet, this function does not initiate auth.

Usage

```
sheets_user()
```

Value

An email address or, if no token has been loaded, NULL.

See Also

gargle::token_userinfo(), gargle::token_email(), gargle::token_tokeninfo()

Examples

```
sheets_user()
```

---

spread_sheet *Spread a data frame of cells into spreadsheet shape*

Description

Reshapes a data frame of cells (presumably the output of sheets_cells()) into another data frame, i.e., puts it back into the shape of the source spreadsheet. This function exists primarily for internal use and for testing. The flagship function read_sheet() is what most users are looking for. It is basically sheets_cells() + spread_sheet().

Usage

```
spread_sheet(
  df,
  col_names = TRUE,
  col_types = NULL,
  na = "",
  trim_ws = TRUE,
  guess_max = min(1000, max(df$row)),
  .name_repair = "unique"
)
```

Arguments

df	A data frame with one row per (nonempty) cell, integer variables row and column (probably referring to location within the spreadsheet), and a list-column cell of SHEET_CELL objects.
col_names	TRUE to use the first row as column names, FALSE to get default names, or a character vector to provide column names directly. If user provides col_types, col_names can have one entry per column or one entry per unskipped column.
col_types	Column types. Either NULL to guess all from the spreadsheet or a string of readr-style shortcodes, with one character or code per column. If exactly one col_type is specified, it is recycled. See Details for more.
na	Character vector of strings to interpret as missing values. By default, blank cells are treated as missing data.

`trim_ws`	Logical. Should leading and trailing whitespace be trimmed from cell contents?
`guess_max`	Maximum number of data rows to use for guessing column types.
`.name_repair`	Handling of column names. By default, googlesheets4 ensures column names are not empty and are unique. There is full support for `.name_repair` as documented in `tibble::tibble()`.

Value

A tibble in the shape of the original spreadsheet, but enforcing user's wishes regarding column names, column types, `NA` strings, and whitespace trimming.

Examples

```
if (sheets_has_token()) {
  df <- sheets_cells(sheets_example("mini-gap"))
  spread_sheet(df)

  # ^^ gets same result as ...
  read_sheet(sheets_example("mini-gap"))
}
```

Index