Package 'googlesheets'

May 7, 2017

Title Manage Google Spreadsheets from R

Version 0.2.2

Description Interact with Google Sheets from R.

URL https://github.com/jennybc/googlesheets

BugReports https://github.com/jennybc/googlesheets/issues

Depends R (>= 3.2.0)

License MIT + file LICENSE

LazyData true

Imports cellranger (>= 1.0.0), dplyr (>= 0.4.2), httr (>= 1.1.0), jsonlite, purrr, readr (>= 0.2.2), stats, stringr, tidyr, utils, xml2 (>= 1.0.0)

Suggests covr, ggplot2, knitr, testthat, rmarkdown, rprojroot

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation no

Author Jennifer Bryan [aut, cre], Joanna Zhao [aut]

Maintainer Jennifer Bryan <jenny@rstudio.com>

Repository CRAN

Date/Publication 2017-05-07 07:13:09 UTC

R topics documented:

`cell-specification` *Specify cells for reading or writing*

Description

If you aren't targetting all the cells in a worksheet, you can request that `googlesheets` limit a read or write operation to a specific rectangle of cells. Any function that offers this flexibility will have a `range` argument. The simplest usage is to specify an Excel-like cell range, such as `range = "D12:F15"` or `range = "R1C12:R6C15"`. The cell rectangle can be specified in various other ways, using helper functions. In all cases, cell range processing is handled by the `cellranger` package, where you can find full documentation for the functions used in the examples below.

See Also

The `cellranger` package has full documentation on cell specification and offers additional functions for manipulating "A1:D10" style spreadsheet ranges. Here are the most relevant:

- `cell_limits`

- cell_rows
- cell_cols
- anchored

See a full list of functions in the cellranger index.

Examples

```
## Not run:
gs_gap() %>% gs_read(ws = 2, range = "A1:D8")
gs_gap() %>% gs_read(ws = "Europe", range = cell_rows(1:4))
gs_gap() %>% gs_read(ws = "Europe", range = cell_rows(100:103),
                     col_names = FALSE)
gs_gap() %>% gs_read(ws = "Africa", range = cell_cols(1:4))
gs_gap() %>% gs_read(ws = "Asia", range = cell_limits(c(1, 5), c(4, NA)))

## End(Not run)
```

---

example-sheets *Examples of Google Sheets*

Description

These functions return information on some Google Sheets we've published to the web for use in examples and testing. For example, function names that start with gs_gap_ refer to a spreadsheet based on the Gapminder data, which you can visit it in the browser:

Usage

```
gs_gap_key()

gs_gap_url()

gs_gap_ws_feed()

gs_gap()

gs_mini_gap_key()

gs_mini_gap_url()

gs_mini_gap_ws_feed()

gs_mini_gap()

gs_ff_key()
```

```
gs_ff_url()

gs_ff_ws_feed()

gs_ff()
```

Details

- Gapminder sheet
- mini Gapminder sheet
- Sheet with numeric formatting and formulas

Value

the key, browser URL, worksheets feed or `googlesheet` object corresponding to one of the example sheets

Functions

- `gs_gap_key`: Gapminder sheet key
- `gs_gap_url`: Gapminder sheet URL
- `gs_gap_ws_feed`: Gapminder sheet worksheets feed
- `gs_gap`: Gapminder sheet as registered googlesheet
- `gs_mini_gap_key`: mini Gapminder sheet key
- `gs_mini_gap_url`: mini Gapminder sheet URL
- `gs_mini_gap_ws_feed`: mini Gapminder sheet worksheets feed
- `gs_mini_gap`: mini Gapminder sheet as registered googlesheet
- `gs_ff_key`: Key to a sheet with numeric formatting and formulas
- `gs_ff_url`: URL for a sheet with numeric formatting and formulas
- `gs_ff_ws_feed`: Worksheets feed for a sheet with numeric formatting and formulas
- `gs_ff`: Registered googlesheet for a sheet with numeric formatting and formulas

Examples

```
## Not run:
gs_gap_key()
gs_gap_url()
browseURL(gs_gap_url())
gs_gap_ws_feed() # not so interesting to a user!
gs_gap()

gs_ff_key()
gs_ff_url()
gs_ff()
gs_browse(gs_ff())

## End(Not run)
```

extract_key_from_url *Extract sheet key from a URL*

Description

Extract a sheet's unique key from a wide variety of URLs, i.e. a browser URL for both old and new Sheets, the "worksheets feed", and other links returned by the Sheets API.

Usage

```
extract_key_from_url(url)
```

Arguments

url	character; a URL associated with a Google Sheet

Examples

```
## Not run:
GAP_URL <- gs_gap_url()
GAP_KEY <- extract_key_from_url(GAP_URL)
gap_ss <- gs_key(GAP_KEY)
gap_ss

## End(Not run)
```

gd_token *Retrieve and report on the current token*

Description

Prints information about the Google token that is in force and returns the token invisibly.

Usage

```
gd_token(verbose = TRUE)

gs_token(verbose = TRUE)
```

Arguments

verbose	logical; do you want informative messages?

Value

an OAuth token object, specifically a Token2.0, invisibly

Examples

```
## Not run:
## load/refresh existing credentials, if available
## otherwise, go to browser for authentication and authorization
gs_auth()

gd_token()

## End(Not run)
```

`gd_user` *Retrieve information about the current Google user*

Description

Retrieve information about the Google user that has authorized googlesheets to call the Drive and Sheets APIs on their behalf. As long as `full = FALSE` (the default), only the most useful subset of the information available from the "about" endpoint of the Drive API is returned. This is also the information exposed in the print method:

Usage

```
gd_user(full = FALSE, verbose = TRUE)

gs_user(full = FALSE, verbose = TRUE)
```

Arguments

`full`	Logical, indicating whether to return selected (`FALSE`, the default) or full (`TRUE`) user information.
`verbose`	logical; do you want informative messages?

Details

- User's display name
- User's email
- Date-time of user info lookup
- User's permission ID
- User's root folder ID

When `full = TRUE`, all information provided by the API is returned.

Value

an object of S3 class 'drive_user', which is just a list

See Also

Other auth functions: gs_auth, gs_deauth

Examples

```
## Not run:
## these are synonyms: gd = Google Drive, gs = Google Sheets
gd_user()
gs_user()

## End(Not run)
```

---

googlesheet *Register a Google Sheet*

Description

The googlesheets package must gather information on a Google Sheet from the API prior to any requests to read or write data. We call this **registering** the sheet and store the result in a googlesheet object. Note this object does not contain any sheet data, but rather contains metadata about the sheet. We populate a googlesheet object with information from the worksheets feed and, if available, also from the spreadsheets feed. Choose from the functions below depending on the type of sheet-identifying input you will provide. Is it a sheet title, key, browser URL, or worksheets feed (another URL, mostly used internally)?

Usage

```
gs_title(x, verbose = TRUE)

gs_key(x, lookup = NULL, visibility = NULL, verbose = TRUE)

gs_url(x, lookup = NULL, visibility = NULL, verbose = TRUE)

gs_ws_feed(x, lookup = NULL, verbose = TRUE)

gs_gs(x, visibility = NULL, verbose = TRUE)
```

Arguments

x	sheet-identifying information; a character vector of length one holding sheet title, key, browser URL or worksheets feed OR, in the case of gs_gs only, a googlesheet object
verbose	logical; do you want informative messages?

`lookup` logical, optional. Controls whether `googlesheets` will place authorized API requests during registration. If unspecified, will be set to `TRUE` if authorization has previously been used in this R session, if working directory contains a file named `.httr-oauth`, or if `x` is a worksheets feed or `googlesheet` object that specifies "public" visibility.

`visibility` character, either "public" or "private". Consulted during explicit construction of a worksheets feed from a key, which happens only when `lookup = FALSE` and `googlesheets` is prevented from looking up information in the spreadsheets feed. If unspecified, will be set to "public" if `lookup = FALSE` and "private" if `lookup = TRUE`. Consult the API docs for more info about visibility

Details

A registered `googlesheet` will contain information on:

- `sheet_key` the key of the spreadsheet
- `sheet_title` the title of the spreadsheet
- `n_ws` the number of worksheets contained in the spreadsheet
- `ws_feed` the "worksheets feed" of the spreadsheet
- `updated` the time of last update (at time of registration)
- `reg_date` the time of registration
- `visibility` visibility of spreadsheet (Google's confusing vocabulary); actually, does not describe a property of spreadsheet itself but rather whether requests will be made with or without authorization
- `is_public` logical indicating visibility is "public" (meaning unauthenticated requests will be sent), as opposed to "private" (meaning authenticated requests will be sent)
- `author` the name of the owner
- `email` the email of the owner
- `links` data.frame of links specific to the spreadsheet
- `ws` a data.frame about the worksheets contained in the spreadsheet

A `googlesheet` object will contain this information from the spreadsheets feed if it was available at the time of registration:

- `alt_key` alternate key; applies only to "old" sheets

Since the spreadsheets feed contains private user data, `googlesheets` must be properly authorized to access it. So a `googlesheet` object will only contain info from the spreadsheets feed if `lookup = TRUE`, which directs us to look up sheet-identifying information in the spreadsheets feed.

Value

a `googlesheet` object

googlesheets googlesheets *package*

Description

Google spreadsheets R API

Details

See the README on CRAN or GitHub

gs_add_row *Append rows to a spreadsheet*

Description

Add rows to an existing worksheet within an existing spreadsheet. This is based on the list feed, which has a strong assumption that the data occupies a neat rectangle in the upper left corner of the sheet. This function specifically uses this method, which "inserts the new row immediately after the last row that appears in the list feed, which is to say immediately before the first entirely blank row."

Usage

```
gs_add_row(ss, ws = 1, input = "", verbose = TRUE)
```

Arguments

ss	a registered Google spreadsheet, i.e. a googlesheet object
ws	positive integer or character string specifying index or title, respectively, of the worksheet
input	new cell values, as an object that can be coerced into a character vector, presumably an atomic vector, a factor, a matrix or a data.frame
verbose	logical; do you want informative messages?

Details

At the moment, this function will only work in a sheet that has a proper header row of variable or column names and at least one pre-existing data row. If you get `Error : No matches`, that suggests the worksheet doesn't meet these minimum requirements. In the future, we will try harder to populate the sheet as necessary, e.g. create default variable names in a header row and be able to cope with `input` being the first row of data.

If `input` is two-dimensional, internally we call `gs_add_row` once per input row.

See Also

gs_edit_cells

Examples

```
## Not run:
yo <- gs_copy(gs_gap(), to = "yo")
yo <- gs_add_row(yo, ws = "Oceania",
                 input = c("Valinor", "Aman", "2015", "10000",
                           "35", "1000.5"))
tail(gs_read(yo, ws = "Oceania"))

gs_delete(yo)

## End(Not run)
```

---

gs_auth *Authorize* googlesheets

Description

Authorize googlesheets to view and manage your files. You will be directed to a web browser, asked to sign in to your Google account, and to grant googlesheets permission to operate on your behalf with Google Sheets and Google Drive. By default, these user credentials are cached in a file named .httr-oauth in the current working directory, from where they can be automatically refreshed, as necessary.

Usage

```
gs_auth(token = NULL, new_user = FALSE,
  key = getOption("googlesheets.client_id"),
  secret = getOption("googlesheets.client_secret"),
  cache = getOption("googlesheets.httr_oauth_cache"), verbose = TRUE)
```

Arguments

token	optional; an actual token object or the path to a valid token stored as an .rds file
new_user	logical, defaults to FALSE. Set to TRUE if you want to wipe the slate clean and re-authenticate with the same or different Google account. This disables the .httr-oauth file in current working directory.
key, secret	the "Client ID" and "Client secret" for the application; defaults to the ID and secret built into the googlesheets package
cache	logical indicating if googlesheets should cache credentials in the default cache file .httr-oauth
verbose	logical; do you want informative messages?

Details

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

- force the creation of a new token
- retrieve current token as an object, for possible storage to an `.rds` file
- read the token from an object or from an `.rds` file
- provide your own app key and secret – this requires setting up a new project in Google Developers Console
- prevent caching of credentials in `.httr-oauth`

In a direct call to `gs_auth`, the user can provide the token, app key and secret explicitly and can dictate whether interactively-obtained credentials will be cached in `.httr_oauth`. If unspecified, these arguments are controlled via options, which, if undefined at the time `googlesheets` is loaded, are defined like so:

key Set to option `googlesheets.client_id`, which defaults to a client ID that ships with the package

secret Set to option `googlesheets.client_secret`, which defaults to a client secret that ships with the package

cache Set to option `googlesheets.httr_oauth_cache`, which defaults to `TRUE`

To override these defaults in persistent way, predefine one or more of them with lines like this in a `.Rprofile` file:

```
options(googlesheets.client_id = "FOO",
        googlesheets.client_secret = "BAR",
        googlesheets.httr_oauth_cache = FALSE)
```

See `Startup` for possible locations for this file and the implications thereof.

More detail is available from Using OAuth 2.0 for Installed Applications. See `gs_webapp_auth_url` and `gs_webapp_get_token` for functions that execute the "web server application" flow.

Value

an OAuth token object, specifically a `Token2.0`, invisibly

See Also

Other auth functions: `gd_user`, `gs_deauth`

Examples

```
## Not run:
## load/refresh existing credentials, if available
## otherwise, go to browser for authentication and authorization
gs_auth()

## force a new token to be obtained
gs_auth(new_user = TRUE)

## store token in an object and then to file
ttt <- gs_auth()
saveRDS(ttt, "ttt.rds")

## load a pre-existing token
gs_auth(token = ttt)       # from an object
gs_auth(token = "ttt.rds") # from .rds file

## End(Not run)
```

---

gs_browse *Visit a Google Sheet in the browser*

Description

Visit a Google Sheet in the browser

Usage

```
gs_browse(ss, ws = 1)
```

Arguments

ss	a registered Google spreadsheet, i.e. a googlesheet object
ws	positive integer or character string specifying index or title, respectively, of the worksheet

Value

the googlesheet object given as input, invisibly

Examples

```
## Not run:
gap_ss <- gs_gap()
gs_browse(gap_ss)
gs_browse(gap_ss, ws = 3)
gs_browse(gap_ss, ws = "Europe")

## assign and browse at once
```

```
gap_ss <- gs_gap() %>% gs_browse()

## End(Not run)
```

---

gs_copy *Copy an existing spreadsheet*

Description

You can copy a spreadsheet that you own or a sheet owned by a third party that has been made accessible via the sharing dialog options. This function calls the Google Drive API.

Usage

```
gs_copy(from, to = NULL, verbose = TRUE)
```

Arguments

from	a registered Google spreadsheet, i.e. a googlesheet object
to	character string giving the new title of the sheet; if NULL, then the copy will be titled "Copy of ..."
verbose	logical; do you want informative messages?

Examples

```
## Not run:
# copy the Gapminder example sheet
gap_ss <- gs_copy(gs_gap(), to = "Gapminder_copy")
gap_ss
gs_delete(gap_ss)

## End(Not run)
```

---

gs_deauth *Suspend authorization*

Description

Suspend googlesheets' authorization to place requests to the Drive and Sheets APIs on behalf of the authenticated user.

Usage

```
gs_deauth(clear_cache = TRUE, verbose = TRUE)
```

Arguments

clear_cache	logical indicating whether to disable the .httr-oauth file in working directory, if such exists, by renaming to .httr-oauth-SUSPENDED
verbose	logical; do you want informative messages?

See Also

Other auth functions: gd_user, gs_auth

Examples

```
## Not run:
gs_deauth()

## End(Not run)
```

---

gs_delete *Delete a spreadsheet*

Description

Move a spreadsheet to trash on Google Drive. You must own a sheet in order to move it to the trash. If you try to delete a sheet you do not own, a 403 Forbidden HTTP status code will be returned; third party spreadsheets can only be moved to the trash manually in the web browser (which only removes them from your Google Sheets home screen, in any case). If you trash a spreadsheet that is shared with others, it will no longer appear in any of their Google Drives. If you delete something by mistake, remain calm, and visit the trash in Google Drive, find the sheet, and restore it.

Usage

```
gs_delete(ss, verbose = TRUE)
```

Arguments

ss	a registered Google spreadsheet, i.e. a googlesheet object
verbose	logical; do you want informative messages?

Value

logical indicating if the deletion was successful

See Also

gs_grepdel and gs_vecdel for handy wrappers that help you delete sheets by title, with the ability to delete multiple sheets at once

Other sheet deletion functions: gs_grepdel

Examples

```
## Not run:
foo <- gs_new("new_sheet")
gs_delete(foo)

## End(Not run)
```

`gs_download` *Download a spreadsheet*

Description

Export a Google Sheet as a .csv, .pdf, or .xlsx file. You can download a sheet that you own or a sheet owned by a third party that has been made accessible via the sharing dialog options. You can download the entire spreadsheet (.pdf and .xlsx formats only) or a single worksheet (all formats). This function calls the Google Drive API.

Usage

```
gs_download(from, ws = NULL, to = NULL, overwrite = FALSE,
  verbose = TRUE)
```

Arguments

`from`	a registered Google spreadsheet, i.e. a `googlesheet` object
`ws`	positive integer or character string specifying index or title, respectively, of the worksheet
`to`	path to write file; file extension must be one of .csv, .pdf, or .xlsx, which dictates the export format; defaults to `foo.xlsx` where `foo` is a safe filename constructed from the title of the Sheet being downloaded
`overwrite`	logical, indicating whether to overwrite an existing local file
`verbose`	logical; do you want informative messages?

Details

If the worksheet is unspecified, i.e. if `ws = NULL`, then the entire spreadsheet will be exported (.pdf and xlsx formats) or the first worksheet will be exported (.csv format)

Value

The normalized path of the downloaded file, after confirmed success, or `NULL`, otherwise, invisibly.

Examples

```
## Not run:
gs_download(gs_gap(), to = "gapminder.xlsx")
file.remove("gapminder.xlsx")

## End(Not run)
```

`gs_edit_cells` *Edit cells*

Description

Modify the contents of one or more cells. The cells to be edited are specified implicitly by a single anchor cell, which will be the upper left corner of the edited cell region, and the size and shape of the input. If the input has rectangular shape, i.e. is a data.frame or matrix, then a similarly shaped range of cells will be updated. If the input has no dimension, i.e. it's a vector, then `byrow` controls whether edited cells will extend from the anchor across a row or down a column.

Usage

```
gs_edit_cells(ss, ws = 1, input = "", anchor = "A1", byrow = FALSE,
  col_names = NULL, trim = FALSE, verbose = TRUE)
```

Arguments

`ss`	a registered Google spreadsheet, i.e. a `googlesheet` object
`ws`	positive integer or character string specifying index or title, respectively, of the worksheet
`input`	new cell values, as an object that can be coerced into a character vector, presumably an atomic vector, a factor, a matrix or a data.frame
`anchor`	single character string specifying the upper left cell of the cell range to edit; positioning notation can be either "A1" or "R1C1"
`byrow`	logical; should we fill cells across a row (`byrow = TRUE`) or down a column (`byrow = FALSE`, default); consulted only when `input` is a vector, i.e. `dim(input)` is `NULL`
`col_names`	logical; indicates whether column names of input should be included in the edit, i.e. prepended to the input; consulted only when `length(dim(input))` equals 2, i.e. `input` is a matrix or data.frame
`trim`	logical; do you want the worksheet extent to be modified to correspond exactly to the cells being edited?
`verbose`	logical; do you want informative messages?

See Also

`gs_add_row`

Examples

```
## Not run:
yo <- gs_new("yo")
yo <- gs_edit_cells(yo, input = head(iris), trim = TRUE)
gs_read(yo)

yo <- gs_ws_new(yo, ws = "byrow_FALSE")
yo <- gs_edit_cells(yo, ws = "byrow_FALSE",
                    input = LETTERS[1:5], anchor = "A8")
gs_read_cellfeed(yo, ws = "byrow_FALSE", range = "A8:A12") %>%
  gs_simplify_cellfeed()

yo <- gs_ws_new(yo, ws = "byrow_TRUE")
yo <- gs_edit_cells(yo, ws = "byrow_TRUE", input = LETTERS[1:5],
                    anchor = "A8", byrow = TRUE)
gs_read_cellfeed(yo, ws = "byrow_TRUE", range = "A8:E8") %>%
  gs_simplify_cellfeed()

yo <- gs_ws_new(yo, ws = "col_names_FALSE")
yo <- gs_edit_cells(yo, ws = "col_names_FALSE", input = head(iris),
                    trim = TRUE, col_names = FALSE)
gs_read_cellfeed(yo, ws = "col_names_FALSE") %>%
  gs_reshape_cellfeed(col_names = FALSE)

gs_delete(yo)

## End(Not run)
```

---

gs_grepdel *Delete several spreadsheets at once by title*

Description

These functions violate the general convention of operating on a registered Google sheet, i.e. on a googlesheet object. But the need to delete a bunch of sheets at once, based on a vector of titles or on a regular expression, came up so much during development and testing, that it seemed wise to package this as a function.

Usage

```
gs_grepdel(regex, ..., verbose = TRUE)

gs_vecdel(vec, verbose = TRUE)
```

Arguments

regex	character; a regular expression; sheets whose titles match will be deleted
...	optional arguments to be passed to grep when matching regex to sheet titles

verbose	logical; do you want informative messages?
vec	character vector of sheet titles to delete

See Also

gs_delete for more detail on what you can and cannot delete and how to recover from accidental deletion

Other sheet deletion functions: gs_delete

Examples

```
## Not run:
sheet_title <- c("cat", "catherine", "tomCAT", "abdicate", "FLYCATCHER")
ss <- lapply(paste0("TEST-", sheet_title), gs_new)
# list, for safety!, then delete 'TEST-abdicate' and 'TEST-catherine'
gs_ls(regex = "TEST-[a-zA-Z]*cat[a-zA-Z]+$")
gs_grepdel(regex = "TEST-[a-zA-Z]*cat[a-zA-Z]+$")

# list, for safety!, then delete the rest,
# i.e. 'TEST-cat', 'TEST-tomCAT', and 'TEST-FLYCATCHER'
gs_ls(regex = "TEST-[a-zA-Z]*cat[a-zA-Z]*$", ignore.case = TRUE)
gs_grepdel(regex = "TEST-[a-zA-Z]*cat[a-zA-Z]*$", ignore.case = TRUE)

## using gs_vecdel()
sheet_title <- c("cat", "catherine", "tomCAT", "abdicate", "FLYCATCHER")
ss <- lapply(paste0("TEST-", sheet_title), gs_new)
# delete two of these sheets
gs_vecdel(c("TEST-cat", "TEST-abdicate"))
# see? they are really gone, but the others remain
gs_ls(regex = "TEST-[a-zA-Z]*cat[a-zA-Z]*$", ignore.case = TRUE)
# delete the remainder
gs_vecdel(c("TEST-FLYCATCHER", "TEST-tomCAT", "TEST-catherine"))
# see? they are all gone now
gs_ls(regex = "TEST-[a-zA-Z]*cat[a-zA-Z]*$", ignore.case = TRUE)

## End(Not run)
```

---

gs_inspect *Visual overview of populated cells*

Description

This function is still experimental. Expect it to change! Or disappear? This function plots a data.frame and gives a sense of what sort of data is where (e.g. character vs. numeric vs factor). Empty cells (ie. NA's) are also indicated. The purpose is to get oriented to sheets that contain more than one data rectangle. Right now, due to the tabular, data-frame nature of the input, we aren't really conveying when disparate data types appear in a column. That might be something to work on in a future version, if this proves useful. That would require working with cell-by-cell data, i.e. from the cell feed.

Usage

```
gs_inspect(x)
```

Arguments

x	data.frame or tbl_df

Value

a ggplot object

Examples

```
## Not run:
gs_inspect(iris)

# data recorded from a game of ultimate frisbee
ulti_key <- "1223dpf3vnjZUYUnCM8rBSig3JlGrAu1Qu6VmPvdEn4M"
ulti_ss <- ulti_key %>% gs_key()
ulti_dat <- ulti_ss %>% gs_read()
gs_inspect(ulti_dat)

# totally synthetic example
x <- suppressWarnings(matrix(0:1, 21, 21))
x[sample(21^2, 10)] <- NA
x <- as.data.frame(x)
some_columns <- seq(from = 1, to = 21, by = 3)
x[some_columns] <- lapply(x[some_columns], as.numeric)
gs_inspect(x)

## End(Not run)
```

---

gs_ls *List sheets a la Google Sheets home screen*

Description

Lists spreadsheets that the user would see in the Google Sheets home screen: https://docs.google.com/spreadsheets/. This function returns the information available from the spreadsheets feed of the Google Sheets API. Since this is non-public user data, use of gs_ls will require authorization

Usage

```
gs_ls(regex = NULL, ..., verbose = TRUE)
```

Arguments

regex	character; one or more regular expressions; if non-NULL only sheets whose titles match will be listed; multiple regular expressions are concatenated with the vertical bar
...	optional arguments to be passed to grep when matching regex to sheet titles
verbose	logical; do you want informative messages?

Details

This listing gives a *partial* view of the sheets available for access (why just partial? see below). For these sheets, we retrieve sheet title, sheet key, author, user's permission, date-time of last update, version (old vs new sheet?), various links, and an alternate key (only relevant to old sheets).

The resulting table provides a map between readily available information, such as sheet title, and more obscure information you might use in scripts, such as the sheet key. This sort of "table lookup" is exploited in the functions gs_title, gs_key, gs_url, and gs_ws_feed, which register a sheet based on various forms of user input.

Which sheets show up in this table? Certainly those owned by the user. But also a subset of the sheets owned by others but visible to the user. We have yet to find explicit Google documentation on this matter. Anecdotally, sheets owned by a third party but for which the user has read access seem to appear in this listing if the user has visited them in the browser. This is an important point for usability because a sheet can be summoned by title instead of key *only* if it appears in this listing. For shared sheets that may not appear in this listing, a more robust workflow is to specify the sheet via its browser URL or unique sheet key.

Value

a googlesheet_ls object, which is a tbl_df with one row per sheet (we use a custom class only to control how this object is printed)

Examples

```
## Not run:
gs_ls()

yo_names <- paste0(c("yo", "YO"), c("", 1:3))
yo_ret <- yo_names %>% lapply(gs_new)
gs_ls("yo")
gs_ls("yo", ignore.case = TRUE)
gs_ls("yo[23]", ignore.case = TRUE)
gs_grepdel("yo", ignore.case = TRUE)
gs_ls("yo", ignore.case = TRUE)

c("foo", "yo") %>% lapply(gs_new)
gs_ls("yo")
gs_ls("yo|foo")
gs_ls(c("foo", "yo"))
gs_vecdel(c("foo", "yo"))
```

```
## End(Not run)
```

---

gs_new *Create a new spreadsheet*

Description

Create a new spreadsheet in your Google Drive. It will contain a single worksheet which, by default, will [1] have 1000 rows and 26 columns, [2] contain no data, and [3] be titled "Sheet1". Use the `ws_title`, `row_extent`, `col_extent`, and `...` arguments to give the worksheet a different title or extent or to populate it with some data. This function calls the Google Drive API to create the sheet and edit the worksheet name or extent. If you provide data for the sheet, then this function also calls the Google Sheets API.

Usage

```
gs_new(title = "my_sheet", ws_title = NULL, row_extent = NULL,
  col_extent = NULL, ..., verbose = TRUE)
```

Arguments

`title`	the title for the new spreadsheet
`ws_title`	the title for the new, sole worksheet; if unspecified, the Google Sheets default is "Sheet1"
`row_extent`	integer for new row extent; if unspecified, the Google Sheets default is 1000
`col_extent`	integer for new column extent; if unspecified, the Google Sheets default is 26
`...`	optional arguments passed along to `gs_edit_cells` in order to populate the new worksheet with data
`verbose`	logical; do you want informative messages?

Details

We anticipate that **if** the user wants to control the extent of the new worksheet, it will be by providing input data and specifying 'trim = TRUE' (see `gs_edit_cells`) or by specifying `row_extent` and `col_extent` directly. But not both ... although we won't stop you. In that case, note that explicit worksheet sizing occurs before data insertion. If data insertion triggers any worksheet resizing, that will override any usage of `row_extent` or `col_extent`.

Value

a `googlesheet` object

See Also

`gs_edit_cells` for specifics on populating the new sheet with some data and `gs_upload` for creating a new spreadsheet by uploading a local file. Note that `gs_upload` is likely much faster than using `gs_new` and/or `gs_edit_cells`, so try both if speed is a concern.

Examples

```
## Not run:
foo <- gs_new()
foo
gs_delete(foo)

foo <- gs_new("foo", ws_title = "numero uno", 4, 15)
foo
gs_delete(foo)

foo <- gs_new("foo", ws = "I know my ABCs", input = letters, trim = TRUE)
foo
gs_delete(foo)

## End(Not run)
```

`gs_read` *Read data*

Description

This function reads data from a worksheet and returns a data frame. It wraps up the most common usage of other, lower-level functions for data consumption and transformation, but you can call always call them directly for finer control.

Usage

```
gs_read(ss, ws = 1, range = NULL, literal = TRUE, ..., verbose = TRUE)
```

Arguments

`ss`	a registered Google spreadsheet, i.e. a `googlesheet` object
`ws`	positive integer or character string specifying index or title, respectively, of the worksheet
`range`	a cell range, as described in `cell-specification`
`literal`	logical, indicating whether to work only with literal values returned by the API or to consult alternate cell contents
`...`	**Optional** arguments to control data download, parsing, and reshaping; for most purposes, the defaults should be fine. Anything that is not listed here will be silently ignored.

`progress` Logical. Whether to display download progress if in an interactive session.

`col_types` Seize control of type conversion for variables. Passed straight through to `readr::read_csv` or `readr::type_convert`. Follow those links or read the `vignette("column-types")` for details.

`locale, trim_ws, na` Specify locale, the fate of leading or trailing whitespace, or a character vector of strings that should become missing values. Passed straight through to `readr::read_csv` or `readr::type_convert`.

`comment, skip, n_max` Specify a string used to identify comments, request to skip lines before reading data, or specify the maximum number of data rows to read.

`col_names` Either `TRUE`, `FALSE` or a character vector of column names. If `TRUE`, the first row of the data rectangle will be used for names. If `FALSE`, column names will be X1, X2, etc. If a character vector, it will be used as column names. If the sheet contains column names and you just don't like them, specify `skip = 1` so they don't show up in your data.

`check.names` Logical. Whether to run column names through `make.names` with `unique = TRUE`, just like `read.table` does. By default, `googlesheets` implements the `readr` data ingest philosophy, which leaves column names "as is", with one exception: data frames returned by `googlesheets` will have a name for each variable, even if we have to create one.

`verbose` logical; do you want informative messages?

Details

If the `range` argument is not specified and `literal = TRUE`, all data will be read via `gs_read_csv`. Don't worry – no intermediate `*.csv` files are written! We just request the data from the Sheets API via the `exportcsv` link.

If the `range` argument is specified or if `literal = FALSE`, data will be read for the targetted cells via `gs_read_cellfeed`, then reshaped and type converted with `gs_reshape_cellfeed`. See `gs_reshape_cellfeed` for details.

Value

a `data.frame` or, if `dplyr` is loaded, a `tbl_df`

See Also

The `cell-specification` topic for more about targetting specific cells.

Other data consumption functions: `gs_read_cellfeed`, `gs_read_csv`, `gs_read_listfeed`, `gs_reshape_cellfeed`, `gs_simplify_cellfeed`

Examples

```
## Not run:
gap_ss <- gs_gap()
oceania_csv <- gs_read(gap_ss, ws = "Oceania")
str(oceania_csv)
oceania_csv

gs_read(gap_ss, ws = "Europe", n_max = 4, col_types = c("cccccc"))

gs_read(gap_ss, ws = "Oceania", range = "A1:C4")
gs_read(gap_ss, ws = "Oceania", range = "R1C1:R4C3")
```

```
gs_read(gap_ss, ws = "Oceania", range = "R2C1:R4C3", col_names = FALSE)
gs_read(gap_ss, ws = "Oceania", range = "R2C5:R4C6",
        col_names = c("thing_one", "thing_two"))
gs_read(gap_ss, ws = "Oceania", range = cell_limits(c(1, 3), c(1, 4)),
        col_names = FALSE)
gs_read(gap_ss, ws = "Oceania", range = cell_rows(1:5))
gs_read(gap_ss, ws = "Oceania", range = cell_cols(4:6))
gs_read(gap_ss, ws = "Oceania", range = cell_cols("A:D"))

ff_ss <- gs_ff() # register example sheet with formulas and formatted nums
gs_read(ff_ss)                 # almost all vars are character
gs_read(ff_ss, literal = FALSE) # more vars are properly numeric

## End(Not run)
```

`gs_read_cellfeed` *Read data from cells*

Description

This function consumes data via the "cell feed", which, as the name suggests, retrieves data cell by cell. Note that the output is a data frame with **one row per cell**. Consult the Google Sheets API documentation for more details about the cell feed.

Usage

```
gs_read_cellfeed(ss, ws = 1, range = NULL, ..., return_empty = FALSE,
  return_links = FALSE, verbose = TRUE)
```

Arguments

`ss`	a registered Google spreadsheet, i.e. a `googlesheet` object
`ws`	positive integer or character string specifying index or title, respectively, of the worksheet
`range`	a cell range, as described in `cell-specification`
`...`	**Optional** arguments to control data download, parsing, and reshaping; for most purposes, the defaults should be fine. Anything that is not listed here will be silently ignored. `progress` Logical. Whether to display download progress if in an interactive session. `col_types` Seize control of type conversion for variables. Passed straight through to `readr::read_csv` or `readr::type_convert`. Follow those links or read the `vignette("column-types")` for details. `locale, trim_ws, na` Specify locale, the fate of leading or trailing whitespace, or a character vector of strings that should become missing values. Passed straight through to `readr::read_csv` or `readr::type_convert`.

`comment, skip, n_max` Specify a string used to identify comments, request to skip lines before reading data, or specify the maximum number of data rows to read.

`col_names` Either `TRUE`, `FALSE` or a character vector of column names. If `TRUE`, the first row of the data rectangle will be used for names. If `FALSE`, column names will be X1, X2, etc. If a character vector, it will be used as column names. If the sheet contains column names and you just don't like them, specify `skip = 1` so they don't show up in your data.

`check.names` Logical. Whether to run column names through `make.names` with `unique = TRUE`, just like `read.table` does. By default, `googlesheets` implements the `readr` data ingest philosophy, which leaves column names "as is", with one exception: data frames returned by `googlesheets` will have a name for each variable, even if we have to create one.

`return_empty` logical; indicates whether to return empty cells

`return_links` logical; indicates whether to return the edit and self links (used internally in cell editing workflow)

`verbose` logical; do you want informative messages?

Details

Use the `range` argument to specify which cells you want to read. See the examples and the help file for the cell specification functions for various ways to limit consumption to, e.g., a rectangle or certain columns. If `range` is specified, the associated cell limits will be checked for internal consistency and compliance with the known extent of the worksheet. If no limits are provided, all cells will be returned but consider that `gs_read_csv` and `gs_read_listfeed` are much faster ways to consume all the data from a rectangular worksheet.

Empty cells, even if "embedded" in a rectangular region of populated cells, are not normally returned by the cell feed. This function won't return them either when `return_empty = FALSE` (default), but will if you set `return_empty = TRUE`.

Value

a `data.frame` or, if `dplyr` is loaded, a `tbl_df`

See Also

`gs_reshape_cellfeed` or `gs_simplify_cellfeed` to perform reshaping or simplification, respectively; `gs_read` is a pre-made wrapper that combines `gs_read_cellfeed` and `gs_reshape_cellfeed`

Other data consumption functions: `gs_read_csv`, `gs_read_listfeed`, `gs_read`, `gs_reshape_cellfeed`, `gs_simplify_cellfeed`

Examples

```
## Not run:
gap_ss <- gs_gap() # register the Gapminder example sheet
col_4_and_above <-
  gs_read_cellfeed(gap_ss, ws = "Asia", range = cell_limits(c(NA, 4)))
col_4_and_above
```

```
gs_reshape_cellfeed(col_4_and_above)

gs_read_cellfeed(gap_ss, range = "A2:F3")

## End(Not run)
```

`gs_read_csv` *Read data via the* `exportcsv` *link*

Description

This function reads all data from a worksheet and returns it as a `tbl_df` or `data.frame`. Don't be spooked by the "csv" thing – the data is NOT actually written to file during this process. Data is read from the "maximal data rectangle", i.e. the rectangle spanned by the maximal row and column extent of the data. By default, empty cells within this rectangle will be assigned `NA`. This is the fastest method of data consumption, so use it as long as you can tolerate the lack of control re: which cells are being read.

Usage

```
gs_read_csv(ss, ws = 1, ..., verbose = TRUE)
```

Arguments

`ss`	a registered Google spreadsheet, i.e. a `googlesheet` object
`ws`	positive integer or character string specifying index or title, respectively, of the worksheet
`...`	**Optional** arguments to control data download, parsing, and reshaping; for most purposes, the defaults should be fine. Anything that is not listed here will be silently ignored.

`progress` Logical. Whether to display download progress if in an interactive session.

`col_types` Seize control of type conversion for variables. Passed straight through to `readr::read_csv` or `readr::type_convert`. Follow those links or read the `vignette("column-types")` for details.

`locale, trim_ws, na` Specify locale, the fate of leading or trailing whitespace, or a character vector of strings that should become missing values. Passed straight through to `readr::read_csv` or `readr::type_convert`.

`comment, skip, n_max` Specify a string used to identify comments, request to skip lines before reading data, or specify the maximum number of data rows to read.

`col_names` Either `TRUE`, `FALSE` or a character vector of column names. If `TRUE`, the first row of the data rectangle will be used for names. If `FALSE`, column names will be X1, X2, etc. If a character vector, it will be used as column names. If the sheet contains column names and you just don't like them, specify `skip = 1` so they don't show up in your data.

check.names Logical. Whether to run column names through make.names with unique = TRUE, just like read.table does. By default, googlesheets implements the readr data ingest philosophy, which leaves column names "as is", with one exception: data frames returned by googlesheets will have a name for each variable, even if we have to create one.

verbose logical; do you want informative messages?

Value

a data.frame or, if dplyr is loaded, a tbl_df

See Also

Other data consumption functions: gs_read_cellfeed, gs_read_listfeed, gs_read, gs_reshape_cellfeed, gs_simplify_cellfeed

Examples

```
## Not run:
gap_ss <- gs_gap() # register the Gapminder example sheet
oceania_csv <- gs_read_csv(gap_ss, ws = "Oceania")
str(oceania_csv)
oceania_csv

## crazy demo of passing args through to readr::read_csv()
oceania_crazy <- gs_read_csv(gap_ss, ws = "Oceania",
  col_names = paste0("Z", 1:6), na = "1962", col_types = "cccccc", skip = 1)
oceania_crazy

## End(Not run)
```

---

gs_read_listfeed *Read data via the "list feed"*

Description

Gets data via the "list feed", which assumes populated cells form a neat rectangle. The list feed consumes data row by row. The first row is assumed to hold variable or column names; it can be empty. The second row is assumed to hold the first data row and, if it is empty, no data will be read and you will get an empty data frame.

Usage

```
gs_read_listfeed(ss, ws = 1, reverse = NULL, orderby = NULL, sq = NULL,
  ..., verbose = TRUE)
```

Arguments

ss	a registered Google spreadsheet, i.e. a googlesheet object
ws	positive integer or character string specifying index or title, respectively, of the worksheet
reverse	logical, optional. Indicates whether to request reverse row order in the actual API call.
orderby	character, optional. Specifies a column to sort on in the actual API call.
sq	character, optional. Provides a structured query for row filtering in the actual API call.
...	**Optional** arguments to control data download, parsing, and reshaping; for most purposes, the defaults should be fine. Anything that is not listed here will be silently ignored.

progress Logical. Whether to display download progress if in an interactive session.

col_types Seize control of type conversion for variables. Passed straight through to readr::read_csv or readr::type_convert. Follow those links or read the vignette("column-types") for details.

locale, trim_ws, na Specify locale, the fate of leading or trailing whitespace, or a character vector of strings that should become missing values. Passed straight through to readr::read_csv or readr::type_convert.

comment, skip, n_max Specify a string used to identify comments, request to skip lines before reading data, or specify the maximum number of data rows to read.

col_names Either TRUE, FALSE or a character vector of column names. If TRUE, the first row of the data rectangle will be used for names. If FALSE, column names will be X1, X2, etc. If a character vector, it will be used as column names. If the sheet contains column names and you just don't like them, specify skip = 1 so they don't show up in your data.

check.names Logical. Whether to run column names through make.names with unique = TRUE, just like read.table does. By default, googlesheets implements the readr data ingest philosophy, which leaves column names "as is", with one exception: data frames returned by googlesheets will have a name for each variable, even if we have to create one.

verbose	logical; do you want informative messages?

Details

The other read functions are generally superior, so use them if you can. However, you may need to use this function if you are dealing with an "old" Google Sheet, which is beyond the reach of gs_read_csv. The list feed also has some ability to sort and filter rows via the API (more below). Consult the Google Sheets API documentation for more details about the list feed.

Value

a data.frame or, if dplyr is loaded, a tbl_df

Column names

For the list feed, and only for the list feed, the Sheets API wants to transform the variable or column names like so: 'The column names are the header values of the worksheet lowercased and with all non-alpha-numeric characters removed. For example, if the cell A1 contains the value "Time 2 Eat!" the column name would be "time2eat".' In `googlesheets`, we do not let this happen and, instead, use the column names "as is", for consistent output across all `gs_read*` functions. If you direct `gs_read_listfeed` to pass query parameters to the actual API call, you must refer to variables using the column names *under this API-enforced transformation*. For example, to order the data by the column with "Time 2 Eat!" in the header row, you must specify `orderby = "time2eat"` in the `gs_read_listfeed` call.

Sorting and filtering via the API

Why on earth would you want to sort and filter via the API instead of in R? Just because you can? It is conceivable there are situations, such as a large spreadsheet, in which it is faster to sort or filter via API. Be sure to refer to variables using the API-transformed column names explained above! It is a known bug that `reverse=true` alone will NOT, in fact, reverse the row order of the result. In our experience, the `reverse` query parameter will only have effect in combination with explicit specification of a column to sort on via `orderby`. The syntax for these queries is apparently undocumented, so keep it simple or bring your spirit of adventure!

See Also

Other data consumption functions: `gs_read_cellfeed`, `gs_read_csv`, `gs_read`, `gs_reshape_cellfeed`, `gs_simplify_cellfeed`

Examples

```
## Not run:
gap_ss <- gs_gap() # register the Gapminder example sheet
oceania_lf <- gs_read_listfeed(gap_ss, ws = "Oceania")
head(oceania_lf, 3)

## do row ordering and filtering in the API call
oceania_fancy <-
  gs_read_listfeed(gap_ss,
                   ws = "Oceania",
                   reverse = TRUE, orderby = "gdppercap",
                   sq = "lifeexp > 79 or year < 1960")
oceania_fancy

## passing args through to readr::type_convert()
oceania_crazy <-
  gs_read_listfeed(gap_ss,
                   ws = "Oceania",
                   col_names = paste0("z", 1:6), skip = 1,
                   col_types = "ccncnn",
                   na = "1962")
oceania_crazy
```

```
## End(Not run)
```

---

gs_rename *Rename a spreadsheet*

Description

Give a spreadsheet a new name. Note that file names are not necessarily unique within a folder on Google Drive.

Usage

```
gs_rename(ss, to, verbose = TRUE)
```

Arguments

ss	a registered Google spreadsheet, i.e. a googlesheet object
to	character string for new title of spreadsheet
verbose	logical; do you want informative messages?

Value

a googlesheet object

Examples

```
## Not run:
ss <- gs_gap() %>% gs_copy(to = "jekyll")
gs_ls("jekyll")                  ## see? it's there
ss <- ss %>% gs_rename("hyde")
gs_ls("hyde")                    ## see? it's got a new name
gs_delete(ss)

## End(Not run)
```

gs_reshape_cellfeed *Reshape data from the "cell feed"*

Description

Reshape data from the "cell feed", put it in a `tbl_df`, and do type conversion. By default, assuming we're working with the same cells, `gs_reshape_cellfeed` should return the same result as other read functions. But when `literal = FALSE`, something different happens: we attempt to deliver cell contents free of any numeric formatting. Try this if numeric formatting of literal values is causing numeric data to come in as character, to be undesirably rounded, or to be otherwise mangled. Remember you can also control type conversion by using `...` to provide arguments to `readr::type_convert`. See the `vignette("formulas-and-formatting")` for more details.

Usage

```
gs_reshape_cellfeed(x, literal = TRUE, ..., verbose = TRUE)
```

Arguments

`x`	a data frame returned by `gs_read_cellfeed`
`literal`	logical, indicating whether to work only with literal values returned by the API or to consult alternate cell contents
`...`	**Optional** arguments to control data download, parsing, and reshaping; for most purposes, the defaults should be fine. Anything that is not listed here will be silently ignored.

`progress` Logical. Whether to display download progress if in an interactive session.

`col_types` Seize control of type conversion for variables. Passed straight through to `readr::read_csv` or `readr::type_convert`. Follow those links or read the `vignette("column-types")` for details.

`locale, trim_ws, na` Specify locale, the fate of leading or trailing whitespace, or a character vector of strings that should become missing values. Passed straight through to `readr::read_csv` or `readr::type_convert`.

`comment, skip, n_max` Specify a string used to identify comments, request to skip lines before reading data, or specify the maximum number of data rows to read.

`col_names` Either `TRUE`, `FALSE` or a character vector of column names. If `TRUE`, the first row of the data rectangle will be used for names. If `FALSE`, column names will be X1, X2, etc. If a character vector, it will be used as column names. If the sheet contains column names and you just don't like them, specify `skip = 1` so they don't show up in your data.

`check.names` Logical. Whether to run column names through `make.names` with `unique = TRUE`, just like `read.table` does. By default, `googlesheets` implements the `readr` data ingest philosophy, which leaves column names "as is", with one exception: data frames returned by `googlesheets` will have a name for each variable, even if we have to create one.

verbose logical; do you want informative messages?

Value

a data.frame or, if dplyr is loaded, a tbl_df

See Also

Other data consumption functions: gs_read_cellfeed, gs_read_csv, gs_read_listfeed, gs_read, gs_simplify_cellfeed

Examples

```
## Not run:
gap_ss <- gs_gap() # register the Gapminder example sheet
gs_read_cellfeed(gap_ss, "Asia", range = cell_rows(1:4))
gs_reshape_cellfeed(gs_read_cellfeed(gap_ss, "Asia", range = cell_rows(1:4)))
gs_reshape_cellfeed(gs_read_cellfeed(gap_ss, "Asia",
                                     range = cell_rows(2:4)),
                    col_names = FALSE)
gs_reshape_cellfeed(gs_read_cellfeed(gap_ss, "Asia",
                                     range = cell_rows(2:4)),
                    col_names = paste0("yo", 1:6))

ff_ss <- gs_ff() # register example sheet with formulas and formatted nums
ff_cf <- gs_read_cellfeed(ff_ss)
gs_reshape_cellfeed(ff_cf) # almost all vars are character
gs_reshape_cellfeed(ff_cf, literal = FALSE) # more vars are numeric

## End(Not run)
```

---

gs_simplify_cellfeed *Simplify data from the "cell feed"*

Description

In some cases, you do not want to convert the data retrieved from the cell feed into a data frame via gs_reshape_cellfeed. Instead, you want the data as an atomic vector. That's what this function does. Note that, unlike gs_reshape_cellfeed, embedded empty cells will NOT necessarily appear in this result. By default, the API does not transmit data for these cells; googlesheets inserts these cells in gs_reshape_cellfeed because it is necessary to give the data rectangular shape. In contrast, empty cells will only appear in the output of gs_simplify_cellfeed if they were already present in the data from the cell feed, i.e. if the original call to gs_read_cellfeed had argument return_empty set to TRUE.

Usage

```
gs_simplify_cellfeed(x, convert = TRUE, literal = TRUE, locale = NULL,
  trim_ws = NULL, na = NULL, notation = c("A1", "R1C1", "none"),
  col_names = NULL)
```

Arguments

x	a data frame returned by gs_read_cellfeed
convert	logical. Indicates whether to attempt to convert the result vector from character to something more appropriate, such as logical, integer, or numeric. If TRUE, result is passed through readr::type_convert; if FALSE, result will be character.
literal	logical, indicating whether to work only with literal values returned by the API or to consult alternate cell contents
locale, trim_ws, na	Optionally, specify locale, the fate of leading or trailing whitespace, or a character vector of strings that should become missing values. Passed straight through to readr::type_convert.
notation	character. The result vector can have names that reflect which cell the data came from; this argument selects between the "A1" and "R1C1" positioning notations. Specify "none" to suppress names.
col_names	if TRUE, the first row of the input will be interpreted as a column name and NOT included in the result; useful when reading a single column or variable.

Value

a vector

See Also

Other data consumption functions: gs_read_cellfeed, gs_read_csv, gs_read_listfeed, gs_read, gs_reshape_cellfeed

Examples

```
## Not run:
gap_ss <- gs_gap() # register the Gapminder example sheet
(gap_cf <- gs_read_cellfeed(gap_ss, range = cell_rows(1)))
gs_simplify_cellfeed(gap_cf)
gs_simplify_cellfeed(gap_cf, notation = "R1C1")

(gap_cf <- gs_read_cellfeed(gap_ss, range = "A1:A10"))
gs_simplify_cellfeed(gap_cf)
gs_simplify_cellfeed(gap_cf, col_names = FALSE)

ff_ss <- gs_ff() # register example sheet with formulas and formatted nums
ff_cf <- gs_read_cellfeed(ff_ss, range = cell_cols(3))
gs_simplify_cellfeed(ff_cf)                  # rounded to 2 digits
gs_simplify_cellfeed(ff_cf, literal = FALSE) # hello, more digits!

## End(Not run)
```

gs_upload *Upload a file and convert it to a Google Sheet*

Description

Google supports the following file types to be converted to a Google spreadsheet: .xls, .xlsx, .csv, .tsv, .txt, .tab, .xlsm, .xlt, .xltx, .xltm, .ods. The newly uploaded file will appear in your Google Sheets home screen. This function calls the Google Drive API.

Usage

```
gs_upload(file, sheet_title = NULL, verbose = TRUE, overwrite = FALSE)
```

Arguments

file	path to the file to upload
sheet_title	the title of the spreadsheet; optional, if not specified then the name of the file will be used
verbose	logical; do you want informative messages?
overwrite	whether to overwrite an existing Sheet with the same title

Examples

```
## Not run:
write.csv(head(iris, 5), "iris.csv", row.names = FALSE)
iris_ss <- gs_upload("iris.csv")
iris_ss
gs_read(iris_ss)
file.remove("iris.csv")
gs_delete(iris_ss)

## End(Not run)
```

gs_webapp_auth_url *Build URL for authentication*

Description

Build the Google URL that googlesheets needs to direct users to in order to authenticate in a Web Server Application. This function is designed for use in Shiny apps. In contrast, the default authorization sequence in googlesheets is appropriate for a user working directly with R on a local computer, where the default handshakes between the local computer and Google work just fine. The first step in the Shiny-based workflow is to form the Google URL where the user can authenticate him or herself with Google. After success, the response, in the form of an authorization code, is sent to the redirect_uri (see below) which gs_webapp_get_token uses to exchange for an access token. This token is then stored in the usual manner for this package and used for subsequent API requests.

Usage

```
gs_webapp_auth_url(client_id = getOption("googlesheets.webapp.client_id"),
  redirect_uri = getOption("googlesheets.webapp.redirect_uri"),
  access_type = "online", approval_prompt = "auto")
```

Arguments

client_id	client id obtained from Google Developers Console
redirect_uri	where the response is sent, should be one of the redirect_uri values listed for the project in Google's Developer Console, must match exactly as listed including any trailing '/'
access_type	either "online" (no refresh token) or "offline" (refresh token), determines whether a refresh token is returned in the response
approval_prompt	either "force" or "auto", determines whether the user is reprompted for consent, If set to "auto", then the user only has to see the consent page once for the first time through the authorization sequence. If set to "force" then user will have to grant consent everytime even if they have previously done so.

Details

That was the good news. The bad news is you'll need to use the Google Developers Console to **obtain your own client ID and secret and declare the** `redirect_uri` **specific to your project**. Inform `googlesheets` of this information by providing as function arguments or by defining these options. For example, you can put lines like this into a Project-specific `.Rprofile` file:

options("googlesheets.webapp.client_id" = MY_CLIENT_ID) options("googlesheets.webapp.client_secret" = MY_CLIENT_SECRET) options("googlesheets.webapp.redirect_uri" = MY_REDIRECT_URI)

Based on Google Developers' guide to Using OAuth2.0 for Web Server Applications.

See Also

`gs_webapp_get_token`

---

gs_webapp_get_token *Exchange authorization code for an access token*

Description

Exchange the authorization code in the URL returned by `gs_webapp_auth_url` to get an access_token. This function plays a role similar to `gs_auth`, but in a Shiny-based workflow: it stores a token object in an internal environment, where it can be retrieved for making calls to the Google Sheets and Drive APIs. Read the documentation for `gs_webapp_auth_url` for more details on OAuth2 within Shiny.

Usage

```
gs_webapp_get_token(auth_code,
  client_id = getOption("googlesheets.webapp.client_id"),
  client_secret = getOption("googlesheets.webapp.client_secret"),
  redirect_uri = getOption("googlesheets.webapp.redirect_uri"))
```

Arguments

auth_code	authorization code returned by Google that appears in URL
client_id	client id obtained from Google Developers Console
client_secret	client secret obtained from Google Developers Console
redirect_uri	where the response is sent, should be one of the redirect_uri values listed for the project in Google's Developer Console, must match exactly as listed including any trailing '/'

See Also

gs_webapp_auth_url

---

gs_ws_delete *Delete a worksheet from a spreadsheet*

Description

The worksheet and all of its contents will be removed from the spreadsheet.

Usage

```
gs_ws_delete(ss, ws = 1, verbose = TRUE)
```

Arguments

ss	a registered Google spreadsheet, i.e. a googlesheet object
ws	positive integer or character string specifying index or title, respectively, of the worksheet
verbose	logical; do you want informative messages?

Value

a googlesheet object

Examples

```
## Not run:
gap_ss <- gs_copy(gs_gap(), to = "gap_copy")
gs_ws_ls(gap_ss)
gap_ss <- gs_ws_new(gap_ss, "new_stuff")
gap_ss <- gs_edit_cells(gap_ss, "new_stuff", input = head(iris), trim = TRUE)
gap_ss
gap_ss <- gs_ws_delete(gap_ss, "new_stuff")
gs_ws_ls(gap_ss)
gap_ss <- gs_ws_delete(gap_ss, ws = 3)
gs_ws_ls(gap_ss)
gs_delete(gap_ss)

## End(Not run)
```

---

gs_ws_ls *List the worksheets in a spreadsheet*

Description

Retrieve the titles of all the worksheets in a googlesheet.

Usage

```
gs_ws_ls(ss)
```

Arguments

ss	a registered Google spreadsheet, i.e. a googlesheet object

Examples

```
## Not run:
gs_ws_ls(gs_gap())

## End(Not run)
```

gs_ws_new *Add a new worksheet within a spreadsheet*

Description

Add a new worksheet to an existing spreadsheet. By default, it will [1] have 1000 rows and 26 columns, [2] contain no data, and [3] be titled "Sheet1". Use the `ws_title`, `row_extent`, `col_extent`, and `...` arguments to give the worksheet a different title or extent or to populate it with some data. This function calls the Google Drive API to create the worksheet and edit its title or extent. If you provide data for the sheet, then this function also calls the Google Sheets API. The title of the new worksheet can not be the same as any existing worksheet in the sheet.

Usage

```
gs_ws_new(ss, ws_title = "Sheet1", row_extent = 1000, col_extent = 26,
  ..., verbose = TRUE)
```

Arguments

`ss`	a registered Google spreadsheet, i.e. a `googlesheet` object
`ws_title`	the title for the new, sole worksheet; if unspecified, the Google Sheets default is "Sheet1"
`row_extent`	integer for new row extent; if unspecified, the Google Sheets default is 1000
`col_extent`	integer for new column extent; if unspecified, the Google Sheets default is 26
`...`	optional arguments passed along to `gs_edit_cells` in order to populate the new worksheet with data
`verbose`	logical; do you want informative messages?

Details

We anticipate that **if** the user wants to control the extent of the new worksheet, it will be by providing input data and specifying 'trim = TRUE' (see `gs_edit_cells`) or by specifying `row_extent` and `col_extent` directly. But not both ... although we won't stop you. In that case, note that explicit worksheet sizing occurs before data insertion. If data insertion triggers any worksheet resizing, that will override any usage of `row_extent` or `col_extent`.

Value

a `googlesheet` object

Examples

```
## Not run:
# get a copy of the Gapminder spreadsheet
gap_ss <- gs_copy(gs_gap(), to = "Gapminder_copy")
gap_ss <- gs_ws_new(gap_ss)
```

```
gap_ss <- gs_ws_delete(gap_ss, ws = "Sheet1")
gap_ss <-
  gs_ws_new(gap_ss, ws_title = "Atlantis", input = head(iris), trim = TRUE)
gap_ss
gs_delete(gap_ss)

## End(Not run)
```

`gs_ws_rename` *Rename a worksheet within a spreadsheet*

Description

Give a worksheet a new title that does not duplicate the title of any existing worksheet within the spreadsheet.

Usage

```
gs_ws_rename(ss, from = 1, to, verbose = TRUE)
```

Arguments

`ss`	a registered Google spreadsheet, i.e. a `googlesheet` object
`from`	positive integer or character string specifying index or title, respectively, of the worksheet
`to`	character string for new title of worksheet
`verbose`	logical; do you want informative messages?

Value

a `googlesheet` object

Note

Since the edit link is used in the PUT request, the version path in the url changes everytime changes are made to the worksheet, hence consecutive function calls using the same edit link from the same sheet object without 'refreshing' it by re-registering results in a HTTP 409 Conflict.

Examples

```
## Not run:
gap_ss <- gs_copy(gs_gap(), to = "gap_copy")
gs_ws_ls(gap_ss)
gap_ss <- gs_ws_rename(gap_ss, from = "Oceania", to = "ANZ")
gs_ws_ls(gap_ss)
gap_ss <- gs_ws_rename(gap_ss, from = 1, to = "I am the first sheet!")
gs_ws_ls(gap_ss)
```

```
gs_delete(gap_ss)

## End(Not run)
```

`print.googlesheet` *Print info about a* `googlesheet` *object*

Description

Display information about a Google spreadsheet that has been registered with `googlesheets`: the title of the spreadsheet, date-time of registration, date-time of last update (at time of registration), visibility, permissions, version, the number of worksheets contained, worksheet titles and extent, and sheet key.

Usage

```
## S3 method for class 'googlesheet'
print(x, ...)
```

Arguments

`x`	`googlesheet` object returned by functions such as `gs_title`, `gs_key`, and friends
`...`	potential further arguments (required for Method/Generic reasons)

Examples

```
## Not run:
foo <- gs_new("foo")
foo
print(foo)

## End(Not run)
```

Index