Package 'googleAuthR'

September 9, 2019

Type Package

Version 1.1.1

Title Authenticate and Create Google APIs

Description Create R functions that interact with OAuth2 Google APIs <https://developers.google.com/apis-explorer/> easily, with auto-refresh and Shiny compatibility.

URL http://code.markedmondson.me/googleAuthR/

BugReports https://github.com/MarkEdmondson1234/googleAuthR/issues

Depends R (>= 3.3.0)

Imports assertthat (>= 0.2.0), digest, gargle (>= 0.3.1), httr (>= 1.4.0), jsonlite (>= 1.6), memoise (>= 1.1.0), rlang, utils

Suggests bigQueryR, covr, devtools (>= 1.12.0), formatR (>= 1.4), googleAnalyticsR, knitr, miniUI (>= 0.1.1), rmarkdown, roxygen2 (>= 5.0.0), shiny (>= 0.13.2), testthat, usethis (>= 1.5.1)

License MIT + file LICENSE

LazyData true

VignetteBuilder knitr

RoxygenNote 6.1.1

NeedsCompilation no

Author Mark Edmondson [aut, cre] (<https://orcid.org/0000-0002-8434-3881>), Jennifer Bryan [ctb], Johann deBoer [ctb], Neal Richardson [ctb], David Kulp [ctb], Joe Cheng [ctb]

Maintainer Mark Edmondson <m@sunholo.com>

Repository CRAN

Date/Publication 2019-09-09 08:00:02 UTC

R topics documented:

gar_api_generator *googleAuthR data fetch function generator*

Description

This function generates other functions for use with Google APIs

Usage

```
gar_api_generator(baseURI, http_header = c("GET", "POST", "PUT",
  "DELETE", "PATCH"), path_args = NULL, pars_args = NULL,
  data_parse_function = NULL, customConfig = NULL,
  simplifyVector = getOption("googleAuthR.jsonlite.simplifyVector"),
  checkTrailingSlash = TRUE)
```

Arguments

baseURI	The stem of the API call.
http_header	Type of http request.
path_args	A named list with name=folder in request URI, value=the function variable.
pars_args	A named list with name=parameter in request URI, value=the function variable.
data_parse_function	A function that takes a request response, parses it and returns the data you need.
customConfig	list of httr options such as use_proxy or add_headers that will be added to the request.
simplifyVector	Passed to fromJSON for response parsing
checkTrailingSlash	Default TRUE will append a trailing slash to baseURI if missing

Details

path_args and **pars_args** add default values to the baseURI. NULL entries are removed. Use "" if you want an empty argument.

You don't need to supply access_token for OAuth2 requests in pars_args, this is dealt with in gar_auth()

Add custom configurations to the request in this syntax: `customConfig = list(httr::add_headers("From" = "mark@example.com")`

Value

A function that can fetch the Google API data you specify

Examples

```
## Not run:
library(googleAuthR)
## change the native googleAuthR scopes to the one needed.
options("googleAuthR.scopes.selected" = "email")

get_email <- function(){
   f <- gar_api_generator("https://openidconnect.googleapis.com/v1/userinfo",
                          "POST",
                          data_parse_function = function(x) x$email,
                          checkTrailingSlash = FALSE)

   f()
   }

To use the above functions:
library(googleAuthR)
# go through authentication flow
gar_auth()
s <- get_email()
s

## End(Not run)
```

`gar_api_page` *Takes a generated API function and lets you page through results*

Description

A helper function to help with the common task of paging through large API results.

Usage

```
gar_api_page(f, page_f = function(x) x$nextLink, page_method = c("url",
  "param", "path", "body"), page_arg = NULL, body_list = NULL)
```

Arguments

`f`	a function created by gar_api_generator
`page_f`	A function that will extract the next page information from `f()`. Should return `NULL` if no paging is required, or the value for `page_arg` if it is.
`page_method`	Method of paging: `url` will fetch by changing the fetch URL; `param` will fetch the next page via a parameter set in `page_arg`; `path` will change a path variable set in `page_arg`
`page_arg`	If `page_method="param"`, you need to set this to the parameter that will change for each API page.

body_list — If page_method="body", you need to set the body that will be used in each API call, including the top level parameter page_arg that will be modified by page_f

Details

The page_f function operates on the object returned from the data_parse_function of the function f

If using page_method="url" then then page_f function needs to return the URL that will fetch the next page of results. The default finds this via x$nextLink. This is the easiest to implement if available and is recommended.

If using page_method = "param", then page_f needs to extract the parameter specified in page_arg that will fetch the next page of the results, or NULL if no more pages are required. e.g. if response is x, page_f should extract the next value for the parameter of page_arg that fetches the next results. It should also return NULL if no (more) paging is necessary. See examples. Remember to add the paging argument (e.g. start-index) to the generated function too, so it can be modified.

Value

A list of the API page responses, that you may need to process further into one object.

Examples

```
## Not run:
# demos the two methods for the same function.
# The example is for the Google Analytics management API,
#  you need to authenticate with that to run them.


# paging by using nextLink that is returned in API response
ga_segment_list1 <- function(){

  # this URL will be modified by using the url_override argument in the generated function
  segs <- gar_api_generator("https://www.googleapis.com/analytics/v3/management/segments",
                            "GET",
                            pars_args = list("max-results"=10),
                            data_parse_function = function(x) x)


  gar_api_page(segs,
               page_method = "url",
               page_f = function(x) x$nextLink)

}

# paging by looking for the next start-index parameter

## start by creating the function that will output the correct start-index
paging_function <- function(x){
  next_entry <- x$startIndex + x$itemsPerPage
```

```
  # we have all results e.g. 1001 > 1000
  if(next_entry > x$totalResults){
    return(NULL)
  }

  next_entry
  }

## remember to add the paging argument (start-index) to the generated function too,
##  so it can be modified.
ga_segment_list2 <- function(){

 segs <- gar_api_generator("https://www.googleapis.com/analytics/v3/management/segments",
                           "GET",
                            pars_args = list("start-index" = 1,
                                             "max-results"=10),
                            data_parse_function = function(x) x)

  gar_api_page(segs,
               page_method = "param",
               page_f = paging_function,
               page_arg = "start-index")

  }


identical(ga_segment_list1(), ga_segment_list2())



## End(Not run)
```

`gar_attach_auto_auth` *Auto Authentication function for use within .onAttach*

Description

To be placed within .onAttach to auto load an authentication file from an environment variable.

Usage

```
gar_attach_auto_auth(required_scopes, environment_var = "GAR_AUTH_FILE")
```

Arguments

`required_scopes` A character vector of minimum required scopes for this API library

environment_var
: The name of the environment variable where the file path to the authentication file is kept

  This function works with gar_auto_auth. It is intended to be placed within the .onAttach hook so that it loads when you load your library.

  For auto-authentication to work, the environment variable needs to hold a file path to an existing auth file such as created via gar_auth or a JSON file file download from the Google API console.

Value

Invisible, used for its side effects of calling auto-authentication.

See Also

Other authentication functions: gar_auth_service, gar_auth, gar_auto_auth, gar_gce_auth, get_google_token, token_exists

Examples

```
## Not run:

.onAttach <- function(libname, pkgname){

 googleAuthR::gar_attach_auto_auth("https://www.googleapis.com/auth/urlshortener", "US_AUTH_FILE")

}

## will only work if you have US_AUTH_FILE environment variable pointing to an auth file location
## .Renviron example
US_AUTH_FILE="/home/mark/auth/urlshortnerauth.json"


## End(Not run)
```

---

gar_auth *Authorize* googleAuthR

Description

Wrapper of token_fetch

Usage

```
gar_auth(token = NULL, email = NULL,
  scopes = getOption("googleAuthR.scopes.selected"),
  app = gar_oauth_app(), cache = gargle::gargle_oauth_cache(),
  use_oob = gargle::gargle_oob_default(), package = "googleAuthR",
  new_user = NULL)
```

Arguments

token	an actual token object or the path to a valid token stored as an .rds file
email	An existing gargle cached email to authenticate with or TRUE to authenticate with the only email available.
scopes	Scope of the request
app	app as specified by gar_auth_configure
cache	Where to store authentication tokens
use_oob	Whther to use OOB browserless authetication
package	The name of the package authenticating
new_user	Deprecated, not used

Value

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

See Also

Other authentication functions: gar_attach_auto_auth, gar_auth_service, gar_auto_auth, gar_gce_auth, get_google_token, token_exists

Examples

```
## Not run:

# sets GCP project to auth through
gar_auth_configure(path="path/to/gcp-client.json")

# starts auth process with defaults
gar_auth()

# switching between auth scopes
# first time new scope manual auth, then auto if supplied email
gar_auth(email = "your@email.com",
         scopes = "https://www.googleapis.com/auth/drive")

# ... query Google Drive functions ...

gar_auth(email = "your@email.com",
         scopes = "https://www.googleapis.com/auth/bigquery")
```

```
# ..query BigQuery functions ...

## End(Not run)
```

---

gar_auth_configure *Edit and view auth configuration*

Description

These functions give more control over and visibility into the auth configuration than [gar_auth()] does. ‘gar_auth_configure()‘ lets the user specify their own: * OAuth app, which is used when obtaining a user token. * API key. If googleAuthR is de-authorized via [gar_deauth()], all requests are sent with an API key in lieu of a token. See the vignette [How to get your own API credentials](https://gargle.r-lib.org/articles/get-api-credentials.html) for more. If the user does not configure these settings, internal defaults are used. ‘gar_oauth_app()‘ and ‘gar_api_key()‘ retrieve the currently configured OAuth app and API key, respectively.

Usage

```
gar_auth_configure(app, path, api_key)

gar_api_key()

gar_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

* ‘gar_auth_configure()‘: An object of R6 class [gargle::AuthState], invisibly. * ‘gar_oauth_app()‘: the current user-configured [httr::oauth_app()]. * ‘gar_api_key()‘: the current user-configured API key.

See Also

Other auth functions: `gar_deauth`

Examples

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

# see and store the current user-configured API key (probaby `NULL`)
(original_api_key <- gar_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 = "123456789.apps.googleusercontent.com",
    secret = "abcdefghijklmnopqrstuvwxyz"
  )
  google_key <- "the-key-I-got-for-a-google-API"
  gar_auth_configure(app = google_app, api_key = google_key)

  # confirm the changes
  gar_oauth_app()
  gar_api_key()
}

## Not run:
## bring your own app via JSON downloaded from Google Developers Console
gar_auth_configure(
  path = "/path/to/the/JSON/you/downloaded/from/google/dev/console.json"
)

## End(Not run)

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

---

gar_auth_js *Shiny JavaScript Google Authorisation [Server Module]*

Description

Shiny Module for use with gar_auth_jsUI

Usage

```
gar_auth_js(...)

googleAuth_js(input, output, session,
  message = "Authenticate with your Google account")
```

Arguments

...	Arguments passed to googleAuth_js
input	shiny input
output	shiny output
session	shiny session
message	The message to show when not authenticated

Details

Call via `shiny::callModule(gar_auth_js,"your_id")`

Value

A httr reactive OAuth2.0 token

---

gar_auth_jsUI *Shiny JavaScript Google Authorisation [UI Module]*

Description

A Javascript Google authorisation flow for Shiny apps.

Usage

```
gar_auth_jsUI(...)

googleAuth_jsUI(id, login_class = "btn btn-primary",
  logout_class = "btn btn-danger", login_text = "Log In",
  logout_text = "Log Out", approval_prompt_force = TRUE,
  scopes = getOption("googleAuthR.scopes.selected", "email"))
```

Arguments

...	Arguments passed to googleAuth_jsUI
id	Shiny id
login_class	CSS class of login button
logout_class	CSS class of logout button
login_text	Text to show on login button
logout_text	Text to show on logout button
approval_prompt_force	Whether to force a login each time
scopes	Set the scopes, minimum needs is "email"

Details

Shiny Module for use with gar_auth_js

Value

Shiny UI

`gar_auth_service` *JSON service account authentication*

Description

As well as OAuth2 authentication, you can authenticate without user interaction via Service accounts. This involves downloading a secret JSON key with the authentication details.

To use, go to your Project in the https://console.developers.google.com/apis/credentials/serviceaccountkey

and select JSON Key type. Save the file to your computer and call it via supplying the file path to the `json_file` parameter.

Navigate to it via: Google Dev Console > Credentials > New credentials > Service account Key > Select service account > Key type = JSON

Usage

```
gar_auth_service(json_file,
  scope = getOption("googleAuthR.scopes.selected"))
```

Arguments

`json_file`	the JSON file downloaded from Google Developer Console
`scope`	Scope of the JSON file auth if needed

Value

(Invisible) Sets authentication token

See Also

https://developers.google.com/identity/protocols/OAuth2ServiceAccount

https://developers.google.com/identity/protocols/OAuth2ServiceAccount

Other authentication functions: `gar_attach_auto_auth`, `gar_auth`, `gar_auto_auth`, `gar_gce_auth`, `get_google_token`, `token_exists`

gar_auto_auth *Perform auto authentication*

Description

This helper function lets you use environment variables to auto-authenticate on package load, intended for calling by gar_attach_auto_auth

Usage

```
gar_auto_auth(required_scopes, no_auto = NULL,
  environment_var = "GAR_AUTH_FILE", new_user = NULL)
```

Arguments

required_scopes	Required scopes needed to authenticate - needs to match at least one
no_auto	If TRUE, ignore auto-authentication settings
environment_var	Name of environment var that contains auth file path
new_user	Deprecated, not used

The authentication file can be a `.httr-oauth` file created via gar_auth or a Google service JSON file downloaded from the Google API credential console, with file extension `.json`.

You can use this in your code to authenticate from a file location specified in file, but it is mainly intended to be called on package load via gar_attach_auto_auth.

`environment_var` This is the name that will be called via Sys.getenv on library load. The environment variable will contain an absolute file path to the location of an authentication file.

Value

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

See Also

Help files for .onAttach

Other authentication functions: `gar_attach_auto_auth`, `gar_auth_service`, `gar_auth`, `gar_gce_auth`, `get_google_token`, `token_exists`

gar_batch *Turn a list of gar_fetch_functions into batch functions*

Description

Turn a list of gar_fetch_functions into batch functions

Usage

```
gar_batch(call_list, ...,
  batch_endpoint = getOption("googleAuthR.batch_endpoint", default =
  "https://www.googleapis.com/batch"))
```

Arguments

call_list	a list of functions from gar_api_generator
...	further arguments passed to the data parse function of f
batch_endpoint	the batch API endpoint to send to

Details

This function will turn all the individual Google API functions into one POST request to /batch.

If you need to pass multiple data parse function arguments its probably best to do it in separate batches to avoid confusion.

Value

A list of the Google API responses

See Also

https://developers.google.com/webmaster-tools/v3/how-tos/batch

Documentation on doing batch requests for the search console API. Other Google APIs are similar.

Walk through API calls changing parameters using gar_batch_walk

Other batch functions: gar_batch_walk

Examples

```
## Not run:

## usually set on package load
options(googleAuthR.batch_endpoint = "https://www.googleapis.com/batch/urlshortener/v1")

## from goo.gl API
shorten_url <- function(url){
  body = list(longUrl = url)
```

```
  f <- gar_api_generator("https://www.googleapis.com/urlshortener/v1/url",
                         "POST",
                          data_parse_function = function(x) x$id)

  f(the_body = body)
}


## from goo.gl API
user_history <- function(){
  f <- gar_api_generator("https://www.googleapis.com/urlshortener/v1/url/history",
                       "GET",
                       data_parse_function = function(x) x$items)

  f()
}

gar_batch(list(shorten_url("http://markedmondson.me"), user_history()))


## End(Not run)
```

---

gar_batch_walk *Walk data through batches*

Description

Convenience function for walking through data in batches

Usage

```
gar_batch_walk(f, walk_vector, gar_pars = NULL, gar_paths = NULL,
  the_body = NULL, pars_walk = NULL, path_walk = NULL,
  body_walk = NULL, batch_size = 10, batch_function = NULL,
  data_frame_output = TRUE, ...,
  batch_endpoint = getOption("googleAuthR.batch_endpoint", default =
  "https://www.googleapis.com/batch"))
```

Arguments

f	a function from gar_api_generator
walk_vector	a vector of the parameter or path to change
gar_pars	a list of parameter arguments for f
gar_paths	a list of path arguments for f
the_body	a list of body arguments for f
pars_walk	a character vector of the parameter(s) to modify for each walk of f
path_walk	a character vector of the path(s) to modify for each walk of f

body_walk	a character vector of the body(s) to modify for each walk of f
batch_size	size of each request to Google /batch API
batch_function	a function that will act on the result list of each batch API call
data_frame_output	if the list of lists are dataframes, you can bind them all by setting to TRUE
...	further arguments passed to the data parse function of f
batch_endpoint	the batch API endpoint to send

Details

You can modify more than one parameter or path arg, but it must be the same walked vector e.g. start = end = x

Many Google APIs have batch_size limits greater than 10, 1000 is common.

The 'f' function needs to be a 'gar_api_generator()' function that uses one of 'path_args', 'pars_args' or 'body_args' to construct the URL (rather than say using 'sprintf()' to create the API URL).

You don't need to set the headers in the Google docs for batching API functions - those are done for you.

The argument 'walk_vector' needs to be a vector of the values of the arguments to walk over, which you indicate will walk over the pars/path or body arguments on the function via on of the '*_walk' arguments e.g. if walking over id=1, id=2, for a path argument then it would be 'path_walk="id"' and 'walk_vector=c(1,2,3,4)'

The 'gar_*' parameter is required to pass intended for other arguments to the function 'f' you may need to pass through.

'gar_batch_walk()' only supports changing one value at a time, for one or multiple arguments (I think only changing the 'start-date', 'end-date' example would be the case when you walk through more than one per call)

'batch_size' should be over 1 for batching to be of any benefit at all

The 'batch_function' argument gives you a way to operate on the parsed output of each call

Value

if data_frame_output is FALSE: A list of lists. Outer list the length of number of batches required, inner lists the results from the calls

if data_frame_output is TRUE: The list of lists will attempt to rbind all the results

See Also

Other batch functions: gar_batch

Examples

```
## Not run:
```

```
# get a webproperty per account
getAccountInfo <- gar_api_generator(
  "https://www.googleapis.com/analytics/v3/management/accounts",
  "GET", data_parse_function = function(x) unique(x$items$id))

getWebpropertyInfo <- gar_api_generator(
 "https://www.googleapis.com/analytics/v3/management/", # don't use sprintf to construct this
  "GET",
  path_args = list(accounts = "default", webproperties = ""),
  data_parse_function = function(x) x$items)

walkData <- function(){

  # here due to R lazy evaluation
  accs <- getAccountInfo()
  gar_batch_walk(getWebpropertyInfo,
                 walk_vector = accs,
                 gar_paths = list("webproperties" = ""),
                 path_walk = "accounts",
                 batch_size = 100, data_frame_output = FALSE)
                 }

# do the walk
walkData()

# to walk body data, be careful to modify a top level body name:
changed_emails <- lapply(email, function(x){userRef = list(email = x)})

batched <- gar_batch_walk(users,
                       walk_vector = changed_emails,
                       the_body = list(
                           permissions = list(
                             local = list(permissions)
                           ),
                           userRef = list(
                             email = email[[1]]
                         )
                        ),
                        body_walk = "userRef",
                        batch_size = 300,
                        data_frame_output = FALSE)

## End(Not run)
```

---

gar_cache_get_loc *Setup where to put cache*

Description

To cache to a file system use `memoise::cache_filesystem("cache_folder")`, suitable for unit testing and works between R sessions.

The cached API calls do not need authentication to be active, but need this function to set caching first.

Usage

```
gar_cache_get_loc()

gar_cache_empty()

gar_cache_setup(mcache = memoise::cache_memory(),
  invalid_func = function(req) {     tryCatch(req$status_code == 200,
  error = function(x) FALSE) })
```

Arguments

`mcache`	A cache method from memoise.
`invalid_func`	A function that takes API response, and returns `TRUE` or `FALSE` whether caching takes place. Default cache everything.

Value

`TRUE` if successful.

Examples

```
## Not run:

# demo function to cache within
shorten_url_cache <- function(url){
  body = list(longUrl = url)
  f <- gar_api_generator("https://www.googleapis.com/urlshortener/v1/url",
                      "POST",
                      data_parse_function = function(x) x)
 f(the_body = body)

 }

 ## only cache if this URL
 gar_cache_setup(invalid_func = function(req){
      req$content$longUrl == "http://code.markedmondson.me/"
 })

 # authentication
 gar_auth()
 ## caches
 shorten_url_cache("http://code.markedmondson.me")
```

```
## read cache
shorten_url("http://code.markedmondson.me")

## ..but dont cache me
shorten_url_cache("http://blahblah.com")


## End(Not run)
```

`gar_check_existing_token` *Check a token vs options*

Description

Useful for debugging authentication issues

Usage

```
gar_check_existing_token(token = .auth$cred)
```

Arguments

`token`	A token to check, default current live session token

Details

Will compare the passed token's settings and compare to set options. If these differ, then reauthentication may be needed.

Value

`FALSE` if the options and current token do not match, `TRUE` if they do.

`gar_create_api_objects` *Create the API objects from the Discovery API*

Description

Create the API objects from the Discovery API

Usage

```
gar_create_api_objects(filename, api_json, format = TRUE)
```

Arguments

filename	File to write the objects to
api_json	The json from gar_discovery_api
format	If TRUE will use tidy_eval on content

Value

TRUE if successful, side-effect creating filename

See Also

Other Google Discovery API functions: gar_create_api_skeleton, gar_create_package, gar_discovery_apis_list, gar_discovery_api

---

gar_create_api_skeleton

Create an API library skeleton

Description

This will create a file with the skeleton of the API functions for the specified library

Usage

```
gar_create_api_skeleton(filename, api_json, format = TRUE)
```

Arguments

filename	R file to write skeleton to
api_json	The json from gar_discovery_api
format	If TRUE will use tidy_eval on content

Value

TRUE if successful, side effect will write a file

See Also

Other Google Discovery API functions: gar_create_api_objects, gar_create_package, gar_discovery_apis_list, gar_discovery_api

`gar_create_package` *Create a Google API package*

Description

Create a Google API package

Usage

```
gar_create_package(api_json, directory, rstudio = TRUE, check = TRUE,
  github = TRUE, format = TRUE, overwrite = TRUE)
```

Arguments

`api_json`	json from gar_discovery_api
`directory`	Where to build the package
`rstudio`	Passed to create_package, creates RStudio project file
`check`	Perform a check on the package once done
`github`	If TRUE will upload package to your github
`format`	If TRUE will use tidy_eval on content
`overwrite`	Whether to overwrite an existing directory if it exists

Details

For github upload to work you need to have your github PAT setup. See use_github.

Uses usethis' usethis to create a package structure then gar_create_api_skeleton and gar_create_api_objects to create starting files for a Google API package.

Value

If check is TRUE, the results of the CRAN check, else FALSE

See Also

https://developers.google.com/discovery/v1/reference/apis/list

A Github repository with 154 R packages generated by this function.

Other Google Discovery API functions: `gar_create_api_objects`, `gar_create_api_skeleton`, `gar_discovery_apis_list`, `gar_discovery_api`

gar_deauth *Suspend authorization*

Description

Put googleAuthR into a de-authorized state. Instead of sending a token, googleAuthR 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 googleAuthR 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 [gar_auth_configure()] and retrieve that key via [gar_api_key()]. In the absence of a user-configured key, a built-in default key is used.

Usage

```
gar_deauth()
```

See Also

Other auth functions: gar_auth_configure

Examples

```
## Not run:
gar_deauth()

## End(Not run)
```

gar_debug_parsing *Read the diagnostic object returned on API parse errors.*

Description

Read the diagnostic object returned on API parse errors.

Usage

```
gar_debug_parsing(filename = "gar_parse_error.rds")
```

Arguments

filename	The file created from API errors, usually called gar_parse_error.rds

Details

When googleAuthR API parsing fails, it will write a file called gar_parse_error.rds to the directory. Feed that file into this function to help diagnose the problem.

gar_discovery_api *Get meta data details for specified Google API*

Description

Does not require authentication

Usage

```
gar_discovery_api(api, version)
```

Arguments

api	The API to fetch
version	The API version to fetch

Value

Details of the API

See Also

https://developers.google.com/discovery/v1/reference/apis/getRest

Other Google Discovery API functions: gar_create_api_objects, gar_create_api_skeleton, gar_create_package, gar_discovery_apis_list

gar_discovery_apis_list *Get a list of Google API libraries*

Description

Does not require authentication

Usage

```
gar_discovery_apis_list()
```

Value

List of Google APIs and their resources

See Also

https://developers.google.com/discovery/v1/reference/apis/list

Other Google Discovery API functions: gar_create_api_objects, gar_create_api_skeleton, gar_create_package, gar_discovery_api

gar_gce_auth *Authenticate on Google Compute Engine*

Description

This takes the metadata auth token in a Google Compute Engine instance as authentication source

Usage

```
gar_gce_auth(service_account = "default",
  scopes = getOption("googleAuthR.scopes.selected"))
```

Arguments

service_account	Specify a different service account from the default
scopes	Scopes for the authentication

Details

service_account is default or the service account email e.g. "service-account-key-json@projectname.iam.gservic

Google Compute Engine instances come with their own authentication tokens.

It has no refresh token so you need to call for a fresh token after approx. one hour. The metadata token will refresh itself when it has about 60 seconds left.

You can only use for scopes specified when creating the instance.

If you want to use them make sure their service account email is added to accounts you want to get data from.

Value

A token

See Also

gar_gce_auth_email

Other authentication functions: gar_attach_auto_auth, gar_auth_service, gar_auth, gar_auto_auth, get_google_token, token_exists

gar_gce_auth_default *Authenticate via gcloud's application-default login*

Description

This allows you to take gcloud's application-default login token and turns it into one that can be used by R

Usage

```
gar_gce_auth_default(scopes = getOption("googleAuthR.scopes.selected",
  "https://www.googleapis.com/auth/cloud-platform"))
```

Arguments

scopes	The scope you created the access_token with

Details

When authenticating on Google Cloud Platform services, if you are using services that take the cloud scopes you can use gar_gce_auth to generate authentication.

However, for other services that require a user login (such as Google Analytics API), you need a method of authentication where you can use your own email login. You have two options - create a token offline and upload it to the instance, or gcloud allows you to generate your own token online via `gcloud auth application-default login && gcloud auth application-default print-access-token`

This function will then take the returned access token and put it within R so it can be used as normal with googleAuthR functions.

See Also

gcloud reference

Examples

```
## Not run:

## in the terminal, issue this gcloud command specifying the scopes to authenticate with
gcloud auth application-default login \
  --scopes=https://www.googleapis.com/auth/analytics.readonly

## access the URL, login and create a verification code, paste in console.

## view then copy-paste the access token, to be passed into the R function
gcloud auth application-default print-access-token

## In R:
gar_gce_auth_default(<token-copy-pasted>,
```

```
    scopes = 'https://www.googleapis.com/auth/analytics.readonly',
    cache_file = 'my_ga.auth')

# use token to authenticate as you would normally with library

## End(Not run)
```

`gar_gce_auth_email` *Get the service email via GCE metadata*

Description

Get the service email via GCE metadata

Usage

```
gar_gce_auth_email(service_account = "default")
```

Arguments

`service_account`	Specify a different service account from the `default` Useful if you don't know the default email and need it for other uses

Value

the email address character string

See Also

gar_gce_auth

`gar_has_token` *Is there a token on hand?*

Description

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

Usage

```
gar_has_token()
```

Value

Logical.

See Also

Other low-level API functions: gar_token

Examples

```
gar_has_token()
```

---

gar_set_client *Setup the clientId, clientSecret and scopes*

Description

Help setup the client ID and secret with the OAuth 2.0 clientID. Do not confuse with Service account keys.

Usage

```
gar_set_client(json = Sys.getenv("GAR_CLIENT_JSON"),
  web_json = Sys.getenv("GAR_CLIENT_WEB_JSON"), scopes = NULL,
  activate = c("offline", "web"))
```

Arguments

json	The file location of an OAuth 2.0 client ID json file
web_json	The file location of client ID json file for web applications
scopes	A character vector of scopes to set
activate	Which credential to activate

Details

This function helps set the options(googleAuthR.client_id), options(googleAuthR.client_secret) and options(googleAuthR.scopes.selected) for you.

You can also set the web application client IDs that are used in Shiny authentication, that are set via the options options(googleAuthR.webapp.client_id), options(googleAuthR.webapp.client_secret)

Note that if you authenticate with a cache token with different values it will overwrite them.

For successful authentication, the API scopes can be browsed via the googleAuthR RStudio addin or the Google API documentation.

Do not confuse this JSON file with the service account keys, that are used to authenticate a service email. This JSON only sets up which app you are going to authenticate with - use gar_auth_service with the Service account keys JSON to perform the actual authentication.

By default the JSON file will be looked for in the location specified by the "GAR_CLIENT_JSON" environment argument, or via "GAR_CLIENT_WEB_JSON" for webapps.

Value

The `project-id` the app has been set for

Author(s)

Idea via @jennybc and @jimhester from `gargle` and `gmailr` libraries.

See Also

https://console.cloud.google.com/apis/credentials

Examples

```
## Not run:

gar_set_client("google-client.json",
               scopes = "http://www.googleapis.com/auth/webmasters")
gar_auth_service("google-service-auth.json")

## End(Not run)
```

---

gar_shiny_auth *Create Authentication within Shiny's server.R*

Description

This can be used at the top of the server function for authentication when you have used gar_shiny_ui to create a login page for your ui function.

In some platforms the URL you are authenticating from will not match the Docker container the script is running in (e.g. shinyapps.io or a kubernetes cluster) - in that case you can manually set it via 'options(googleAuthR.redirect = http://your-shiny-url'). In other circumstances the Shiny app should be able to detect this itself.

Usage

```
gar_shiny_auth(session)
```

Arguments

`session`	Shiny session argument

Details

If using gar_shiny_ui, put this at the top of your server.R function

Author(s)

Based on a gist by Joe Cheng, RStudio

See Also

Other pre-load shiny authentication: gar_shiny_auth_url, gar_shiny_login_ui, gar_shiny_ui, silent_auth

Examples

```
## Not run:
library(shiny)
library(googleAuthR)
gar_set_client()

fileSearch <- function(query) {
  googleAuthR::gar_api_generator("https://www.googleapis.com/drive/v3/files/",
                                 "GET",
                                 pars_args=list(q=query),
                                 data_parse_function = function(x) x$files)()
}

## ui.R
ui <- fluidPage(title = "googleAuthR Shiny Demo",
                textInput("query",
                label = "Google Drive query",
                value = "mimeType != 'application/vnd.google-apps.folder'"),
                tableOutput("gdrive")
                )

## server.R
server <- function(input, output, session){

# this is not reactive, no need as you only reach here authenticated
gar_shiny_auth(session)

output$gdrive <- renderTable({
  req(input$query)

  # no need for with_shiny()
  fileSearch(input$query)

  })
  }

# gar_shiny_ui() needs to wrap the ui you have created above.
shinyApp(gar_shiny_ui(ui), server)

## End(Not run)
```

---

gar_shiny_auth_url *Make a Google Authorisation URL for Shiny*

Description

Set this within your login_ui where you need the Google login.

Usage

```
gar_shiny_auth_url(req, state = getOption("googleAuthR.securitycode"),
  client.id = getOption("googleAuthR.webapp.client_id"),
  client.secret = getOption("googleAuthR.webapp.client_secret"),
  scope = getOption("googleAuthR.scopes.selected"),
  access_type = c("online", "offline"), approval_prompt = c("auto",
  "force"))
```

Arguments

req	a Rook request, do not set as this will be used by Shiny to generate URL
state	URL state
client.id	client.id
client.secret	client.secret
scope	API scopes
access_type	whether to keep the token
approval_prompt	Auto-login if user is recognised or always force signin

See Also

Other pre-load shiny authentication: gar_shiny_auth, gar_shiny_login_ui, gar_shiny_ui, silent_auth

---

gar_shiny_login_ui *A login page for Shiny*

Description

An alternative to the immediate login provided by default by gar_shiny_ui

Usage

```
gar_shiny_login_ui(req, title = "googleAuthR Login Demo")
```

Arguments

req	Passed to gar_shiny_auth_url to generate login URL
title	The title of the page

Details

Use gar_shiny_auth_url to create the login URL. You must leave the first argument free as this is used to generate the login, but you can pass other arguments to customise your UI.

See Also

Other pre-load shiny authentication: gar_shiny_auth_url, gar_shiny_auth, gar_shiny_ui, silent_auth

---

gar_shiny_ui *Create a Google login before your Shiny UI launches*

Description

A function that will turn your ui object into one that will look for Google authentication before loading the main app. Use together with gar_shiny_auth

Usage

```
gar_shiny_ui(ui, login_ui = silent_auth)
```

Arguments

ui	A Shiny ui object
login_ui	A UI or HTML template that is seen before the main app and contains a login in link generated by gar_shiny_auth_url

Details

Put this at the bottom of your ui.R or pass into shinyApp wrapping your created ui.

Author(s)

Based on a gist by Joe Cheng, RStudio

See Also

Other pre-load shiny authentication: gar_shiny_auth_url, gar_shiny_auth, gar_shiny_login_ui, silent_auth

Examples

```
## Not run:
library(shiny)
library(googleAuthR)
gar_set_client()

fileSearch <- function(query) {
```

```
  googleAuthR::gar_api_generator("https://www.googleapis.com/drive/v3/files/",
                                 "GET",
                                 pars_args=list(q=query),
                                 data_parse_function = function(x) x$files)()
}

## ui.R
ui <- fluidPage(title = "googleAuthR Shiny Demo",
                textInput("query",
                label = "Google Drive query",
                value = "mimeType != 'application/vnd.google-apps.folder'"),
                tableOutput("gdrive")
                )

## server.R
server <- function(input, output, session){

# this is not reactive, no need as you only reach here authenticated
gar_shiny_auth(session)

output$gdrive <- renderTable({
  req(input$query)

  # no need for with_shiny()
  fileSearch(input$query)

  })
  }

# gar_shiny_ui() needs to wrap the ui you have created above.
shinyApp(gar_shiny_ui(ui), server)

## End(Not run)
```

---

gar_token *Produce configured token*

Description

For internal use or for those programming around the Google 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, [gar_auth()] is what they need. If there is no current token, [gar_auth()] is called to either load from cache or initiate OAuth2.0 flow. If auth has been deactivated via [gar_deauth()], ‘gar_token()‘ returns ‘NULL‘.

Usage

```
gar_token()
```

Value

A ‘request‘ object (an S3 class provided by [httr][httr::httr]).

See Also

Other low-level API functions: gar_has_token

Examples

```
## Not run:
req <- request_generate(
  "drive.files.get",
  list(fileId = "abc"),
  token = gar_token()
)
req

## End(Not run)
```

---

gar_token_info *Get current token summary*

Description

Get details on the current active auth token to help debug issues

Usage

```
gar_token_info(detail_level = getOption("googleAuthR.verbose", default =
  3))
```

Arguments

detail_level	How much info to show

---

googleAuth *Shiny Google Authorisation [Server Module]*

Description

Server part of shiny module, use with googleAuthUI

Usage

```
googleAuth(input, output, session, login_text = "Login via Google",
  logout_text = "Logout", login_class = "btn btn-primary",
  logout_class = "btn btn-default", access_type = c("online",
  "offline"), approval_prompt = c("auto", "force"), revoke = FALSE)
```

Arguments

input	shiny input
output	shiny output
session	shiny session
login_text	What the login text will read on the button
logout_text	What the logout text will read on the button
login_class	The CSS class for the login link
logout_class	The CSS class for the logout link
access_type	Online or offline access for the authentication URL
approval_prompt	Whether to show the consent screen on authentication
revoke	If TRUE a user on logout will need to re-authenticate

Details

Call via `shiny::callModule(googleAuth,"your_ui_name",login_text = "Login")`

In some platforms the URL you are authenticating from will not match the Docker container the script is running in (e.g. shinyapps.io or a kubernetes cluster) - in that case you can manually set it via 'options(googleAuthR.redirect = http://your-shiny-url'). In other circumstances the Shiny app should be able to detect this itself.

Value

A reactive authentication token

See Also

Other shiny module functions: googleAuthUI

Examples

```
## Not run:
options("googleAuthR.scopes.selected" =
  c("https://www.googleapis.com/auth/urlshortener"))

shorten_url <- function(url){
  body = list(
    longUrl = url
 )

 f <-
   gar_api_generator("https://www.googleapis.com/urlshortener/v1/url",
                     "POST",
                     data_parse_function = function(x) x$id)

 f(the_body = body)
```

```
 }

server <- function(input, output, session){

  ## Create access token and render login button
  access_token <- callModule(googleAuth,
                             "loginButton",
                             login_text = "Login1")

  short_url_output <- eventReactive(input$submit, {
    ## wrap existing function with_shiny
    ## pass the reactive token in shiny_access_token
    ## pass other named arguments
    with_shiny(f = shorten_url,
               shiny_access_token = access_token(),
               url=input$url)
  })

  output$short_url <- renderText({

    short_url_output()

  })

}

## ui
ui <- fluidPage(
  googleAuthUI("loginButton"),
  textInput("url", "Enter URL"),
  actionButton("submit", "Shorten URL"),
  textOutput("short_url")
)

shinyApp(ui = ui, server = server)

## End(Not run)
```

---

googleAuthR *googleAuthR: Easy Authentication with Google OAuth2 APIs*

Description

Get more details on the googleAuthR website.

Default options

These are the default options that you can override via `options()`

- `googleAuthR.batch_endpoint = "https://www.googleapis.com/batch"`
- `googleAuthR.rawResponse = FALSE`
- `googleAuthR.httr_oauth_cache = ".httr-oauth"`
- `googleAuthR.verbose = 3`
- `googleAuthR.client_id = NULL`
- `googleAuthR.client_secret = NULL`
- `googleAuthR.webapp.client_id = NULL`
- `googleAuthR.webapp.client_secret = NULL`
- `googleAuthR.webapp.port = 1221`
- `googleAuthR.jsonlite.simplifyVector = TRUE`
- `googleAuthR.scopes.selected = NULL`
- `googleAuthR.ok_content_types=c("application/json; charset=UTF-8",("text/html; charset=UTF-8"))`
- `googleAuthR.securitycode = paste0(sample(c(1:9,LETTERS,letters),20,replace = T),collapse='')`
- `googleAuthR.tryAttempts = 5`

`googleAuthUI` *Shiny Google Authorisation [UI Module]*

Description

UI part of shiny module, use with googleAuth

Usage

```
googleAuthUI(id)
```

Arguments

`id`	shiny id

Value

A shiny UI for logging in

See Also

Other shiny module functions: `googleAuth`

googleSignIn *Google SignIn [Server Module]*

Description

Shiny Module for use with googleSignInUI. Use when you don't need to call APIs, but would like a login to Shiny.

Usage

```
googleSignIn(input, output, session)
```

Arguments

input	shiny input
output	shiny output (id, name, email, image, signed_in)
session	shiny session

Details

Call via `shiny::callModule(googleSignIn,"your_id")`.

Value

A reactive list with values $g_id, $g_name, $g_email, $g_image and $signed_in.

Author(s)

Based on original code by David Kulp

googleSignInUI *Google SignIn [UI Module]*

Description

Shiny Module for use with googleSignIn. If you just want a login to a Shiny app, without API tokens.

Usage

```
googleSignInUI(id, logout_name = "Sign Out",
  logout_class = "btn-danger")
```

Arguments

id	Shiny id.
logout_name	Character. Custom name of the logout button.
logout_class	Character. Bootstrap class name for buttons, e.g. "btn-info", "btn-dark".

Value

Shiny UI

Author(s)

Based on original code by David Kulp

See Also

https://github.com/dkulp2/Google-Sign-In

---

silent_auth *Silent auth*

Description

The default for logging in via gar_shiny_ui, this creates no login page and just takes you straight to authentication on Shiny app load.

Usage

```
silent_auth(req)
```

Arguments

req	What Shiny uses to check the URL parameters

See Also

Other pre-load shiny authentication: gar_shiny_auth_url, gar_shiny_auth, gar_shiny_login_ui, gar_shiny_ui

skip_if_no_env_auth *Skip test if not authenticated*

Description

Use within tests to skip if a local authentication file isn't available through an environment variable.

Usage

```
skip_if_no_env_auth(env_arg)
```

Arguments

env_arg	The name of the environment argument pointing to the auth file

with_shiny *Turn a googleAuthR data fetch function into a Shiny compatible one*

Description

Turn a googleAuthR data fetch function into a Shiny compatible one

Usage

```
with_shiny(f, shiny_access_token = NULL, ...)
```

Arguments

f	A function generated by googleAuth_fetch_generator.
shiny_access_token	A reactive object that resolves to a token.
...	Other arguments passed to f.

Value

the function f with an extra parameter, shiny_access_token=NULL.

Examples

```
## Not run:
## in global.R

## create the API call function, example with goo.gl URL shortner
library(googleAuthR)
options("googleAuthR.scopes.selected" = c("https://www.googleapis.com/auth/urlshortener"))

shorten_url <- function(url){

  body = list(
    longUrl = url
 )

 f <- gar_api_generator("https://www.googleapis.com/urlshortener/v1/url",
                        "POST",
                        data_parse_function = function(x) x$id)

 f(the_body = body)

 }


## in server.R
library(shiny)
library(googleAuthR)
source('global.R')

shinyServer(function(input, output, session)){

  ## Get auth code from return URL
  access_token  <- reactiveAccessToken(session)

  ## Make a loginButton to display using loginOutput
  output$loginButton <- renderLogin(session, access_token())

  short_url_output <- eventReactive(input$submit, {
  ## wrap existing function with_shiny
  ## pass the reactive token in shiny_access_token
  ## pass other named arguments
    short_url <- with_shiny(f = shorten_url,
                            shiny_access_token = access_token(),
                            url=input$url)

   })

   output$short_url <- renderText({

     short_url_output()

   })
 }
```

```
## in ui.R
library(shiny)
library(googleAuthR)

shinyUI(
  fluidPage(
    loginOutput("loginButton"),
    textInput("url", "Enter URL"),
    actionButton("submit", "Shorten URL"),
    textOutput("short_url")
    ))

## End(Not run)
```

Index