Package ‘googleAuthR’

May 22, 2024

Type Package

Version 2.0.2

Title Authenticate and Create Google APIs

Description Create R functions that interact with OAuth2 Google APIs

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

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

Depends R (>= 3.3.0)

Imports assertthat (>= 0.2.0), cli (>= 2.0.2), digest, gargle (>= 1.2.0), htr (>= 1.4.0), jsonlite (>= 1.6), memoise (>= 1.1.0), rlang, utils

Suggests covr, devtools (>= 1.12.0), formatR (>= 1.4), googleAnalyticsR, knitr, miniUI (>= 0.1.1), rmarkdown, roxygen2 (>= 5.0.0), rstudioapi, shiny (>= 0.13.2), testthat, usethis (>= 1.6.0)

License MIT + file LICENSE

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.1

NeedsCompilation no

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

Maintainer Erik Grönroos <erik.gronroos@8-bit-sheep.com>

Repository CRAN

Date/Publication 2024-05-22 21:20:17 UTC
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>gar_api_generator</td>
<td>3</td>
</tr>
<tr>
<td>gar_api_page</td>
<td>4</td>
</tr>
<tr>
<td>gar_attach_auto_auth</td>
<td>6</td>
</tr>
<tr>
<td>gar_auth</td>
<td>7</td>
</tr>
<tr>
<td>gar_auth_configure</td>
<td>9</td>
</tr>
<tr>
<td>gar_auth_service</td>
<td>10</td>
</tr>
<tr>
<td>gar_auto_auth</td>
<td>11</td>
</tr>
<tr>
<td>gar_batch</td>
<td>12</td>
</tr>
<tr>
<td>gar_batch_walk</td>
<td>13</td>
</tr>
<tr>
<td>gar_cache_get_loc</td>
<td>16</td>
</tr>
<tr>
<td>gar_check_existing_token</td>
<td>17</td>
</tr>
<tr>
<td>gar_create_api_objects</td>
<td>18</td>
</tr>
<tr>
<td>gar_create_apiSkeleton</td>
<td>19</td>
</tr>
<tr>
<td>gar_create_package</td>
<td>19</td>
</tr>
<tr>
<td>gar_deauth</td>
<td>20</td>
</tr>
<tr>
<td>gar_debug_parsing</td>
<td>21</td>
</tr>
<tr>
<td>gar_discovery_api</td>
<td>21</td>
</tr>
<tr>
<td>gar_discovery_apis_list</td>
<td>22</td>
</tr>
<tr>
<td>gar_gce_auth</td>
<td>22</td>
</tr>
<tr>
<td>gar_gce_auth_default</td>
<td>23</td>
</tr>
<tr>
<td>gar_gce_auth_email</td>
<td>24</td>
</tr>
<tr>
<td>gar_has_token</td>
<td>25</td>
</tr>
<tr>
<td>gar_scope_config</td>
<td>26</td>
</tr>
<tr>
<td>gar_service_create</td>
<td>26</td>
</tr>
<tr>
<td>gar_service_provision</td>
<td>28</td>
</tr>
<tr>
<td>gar_setup_auth_check</td>
<td>29</td>
</tr>
<tr>
<td>gar_setup_auth_key</td>
<td>30</td>
</tr>
<tr>
<td>gar_setup_clientid</td>
<td>31</td>
</tr>
<tr>
<td>gar_setup_edit_reviron</td>
<td>31</td>
</tr>
<tr>
<td>gar_setup_env_check</td>
<td>32</td>
</tr>
<tr>
<td>gar_setup_get_authenv</td>
<td>33</td>
</tr>
<tr>
<td>gar_setup_menu</td>
<td>33</td>
</tr>
<tr>
<td>gar_setup_menu_do</td>
<td>34</td>
</tr>
<tr>
<td>gar_set_client</td>
<td>35</td>
</tr>
<tr>
<td>gar_shiny_auth</td>
<td>37</td>
</tr>
<tr>
<td>gar_shiny_auth_url</td>
<td>38</td>
</tr>
<tr>
<td>gar_shiny_login_ui</td>
<td>39</td>
</tr>
<tr>
<td>gar_shiny_ui</td>
<td>40</td>
</tr>
<tr>
<td>gar_token</td>
<td>41</td>
</tr>
<tr>
<td>gar_token_info</td>
<td>42</td>
</tr>
<tr>
<td>googleAuthR</td>
<td>42</td>
</tr>
<tr>
<td>googleSignIn</td>
<td>43</td>
</tr>
<tr>
<td>googleSignInUI</td>
<td>44</td>
</tr>
<tr>
<td>silent_auth</td>
<td>45</td>
</tr>
<tr>
<td>skip_if_no_env_auth</td>
<td>45</td>
</tr>
</tbody>
</table>

Index 46
**gar_api_generator**

---

**googleAuthR data fetch function generator**

---

**Description**

This function generates other functions for use with Google APIs.

**Usage**

```r
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

```r
## 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

```

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

```r
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.
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.

If page_method="param", you need to set this to the parameter that will change for each API page.

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.

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

If using page_method="url" 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.

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

```r
# 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
g_segment_list1 <- function(){
  # this URL will be modified by using the url_override argument in the generated function
                           "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

```r
gpaging_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.

```r
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")
}
```

```r
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

```r
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

evironment_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(), gar_auth_service(), gar_auto_auth(), gar_gce_auth(), get_google_token(), should_skip_token_checks(), 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)
Usage

```r
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"
)
```

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**: Whether to use OOB browserless authentication
- **package**: The name of the package authenticating

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()`, `should_skip_token_checks()`, `token_exists()`

Examples

```r
## 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_configure

```r
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 client, 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("get-api-credentials", package = "gargle")` for more. If the user does not configure these settings, internal defaults are used.

`gar_oauth_client()` and `gar_api_key()` retrieve the currently configured OAuth client and API key, respectively.

**Usage**

```r
gar_auth_configure(app, path, api_key)
gar_api_key()
gar_oauth_app()
```

**Arguments**

- **app**
  A Google OAuth client, presumably constructed via `gargle_oauth_client_from_json`. Note, however, that it is preferred to specify the client with JSON, using the ‘path’ argument.

- **path**
  JSON downloaded from Google Cloud Console, containing a client id and secret, in one of the forms supported for the `txt` argument of `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_client()`’: the current user-configured OAuth client. *‘`gar_api_key()`’: the current user-configured API key.
See Also

Other auth functions: `gar_deauth()`

Examples

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

# see and store the current user-configured API key (probably '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()
}
```

```r
## Not run:
## bring your own app via JSON downloaded from Google Developers Console
## End(Not run)
## restore original auth config
## End(Not run)

```

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(), should_skip_token_checks(), token_exists()

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, environment_var = "GAR_AUTH_FILE")

Arguments

required_scopes Required scopes needed to authenticate - needs to match at least one
environment_var Name of environment var that contains auth file path

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(), gar_auth_service(), gar_gce_auth(), get_google_token(), should_skip_token_checks(), token_exists()

---

gar_batch

\textit{Turn a list of gar_fetch_functions into batch functions}

Description

Turn a list of gar_fetch_functions into batch functions

Usage

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

Arguments

function_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

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:

```r
## 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

```r
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
- `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

```r
## Not run:

# get a webproperty per account
getAccountInfo <- gar_api_generator(
  "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)
  )
)
```
### 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`

```r
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

```r
## 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_api()`, `gar_discovery_apis_list()`
**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**

```r
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_api()`, `gar_discovery_apis_list()`

---

**gar_create_package**

*Create a Google API package*

**Description**

Create a Google API package

**Usage**

```r
gar_create_package(
    api_json,
    directory,
    rstudio = TRUE,
    check = FALSE,
    github = FALSE,
    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 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](https://developers.google.com/discovery/v1/reference/apis/list)
  A Github repository with 154 R packages examples generated by this function.
  Other Google Discovery API functions: `gar_create_api_objects()`, `gar_create_api_skeleton()`, `gar_discovery_api()`, `gar_discovery_apis_list()`

---

**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

```r
gar_deauth()
```

See Also

Other auth functions: `gar_auth_configure()`
**gar_debug_parsing**

Read the diagnostic object returned on API parse errors.

### Description

Read the diagnostic object returned on API parse errors.

### Usage

```r
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

Download the discovery document for an API

### Usage

```r
gar_discovery_api(api, version, a_url = NULL)
```

### Arguments

- **api**
  
  The API to fetch

- **version**
  
  The API version to fetch

- **a_url**
  
  Supply your own discovery URL, for private APIs only

### Value

Details of the API
See Also

https://developers.google.com/discovery/v1/getting_started

Other Google Discovery API functions: gar_create_api_objects(), gar_create_api_skeleton(), gar_create_package(), gar_discovery_apis_list()
Details

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

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.

Use options(gargle.gce.use_ip = TRUE) to activate this upon kubernetes for instance using federated identity

Value

A token

See Also

\texttt{gar\_gce\_auth\_email}

Other authentication functions: \texttt{gar\_attach\_auto\_auth()}, \texttt{gar\_auth()}, \texttt{gar\_auth\_service()}, \texttt{gar\_auto\_auth()}, \texttt{get\_google\_token()}, \texttt{should\_skip\_token\_checks()}, \texttt{token\_exists()}

---

\texttt{gar\_gce\_auth\_default}  \texttt{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

\texttt{gar\_gce\_auth\_default(}
\texttt{  scopes = getOption("googleAuthR.scopes.selected",}
\texttt{      "https://www.googleapis.com/auth/cloud-platform")}
\texttt{)}

Arguments

\texttt{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:

```r
## 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")`
**gar_has_token**

**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`

---

**Description**

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

**Usage**

```r
gar_has_token()
```

**Value**

Logical.

**See Also**

Other low-level API functions: `gar_token()`

**Examples**

```r
gar_has_token()
```
gar_scope_config    Create or add scopes to configuration

Description
Helper for working with scopes

Usage
gar_scope_config(required_scopes)

Arguments
required_scopes
character vector of scopes to add

gar_service_create    Work with service accounts via the API

Description
These functions let you create a service JSON key from an OAuth2 login. You can then assign it roles and do a one time download of a service account key to use for authentication in other Google APIs

Usage
gar_service_create(
    accountId,
    projectId,
    serviceName = "googleAuthR::gar_service_create",
    serviceDescription = "A service account created via googleAuthR"
)

gar_service_grant_roles(
    accountIds,
    roles,
    projectId,
    type = c("serviceAccount", "user", "group")
)

gar_service_get_roles(
    projectId,
    accountId = NULL,
    type = c("serviceAccount", "user", "group"))
gar_service_create

)

gar_service_key(
    accountId,
    projectId,
    file = paste0(accountId, "-auth-key.json")
)

gar_service_key_list(accountId, projectId)

gar_service_list(projectId)

gar_service_get(accountId, projectId)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountId</td>
<td>The service accountId</td>
</tr>
<tr>
<td>projectId</td>
<td>The projectId containing the service account</td>
</tr>
<tr>
<td>serviceName</td>
<td>Name of service account</td>
</tr>
<tr>
<td>serviceDescription</td>
<td>Description of service account</td>
</tr>
<tr>
<td>accountIds</td>
<td>A vector of accountIds in the form <a href="mailto:accountId@projectid.iam.gserviceaccount.com">accountId@projectid.iam.gserviceaccount.com</a></td>
</tr>
<tr>
<td>roles</td>
<td>A character vector of roles to give the accountIds e.g. roles/editor - see list of roles here <a href="https://cloud.google.com/iam/docs/understanding-roles#predefined_roles">https://cloud.google.com/iam/docs/understanding-roles#predefined_roles</a> or in your GCP console <a href="https://console.cloud.google.com/iam-admin/roles/details/roles">https://console.cloud.google.com/iam-admin/roles/details/roles</a></td>
</tr>
<tr>
<td>type</td>
<td>The type of accountId to add role for - e.g. user:<a href="mailto:mark@me.com">mark@me.com</a> or serviceAccount:<a href="mailto:accountId@projectid.iam.gserviceaccount.com">accountId@projectid.iam.gserviceaccount.com</a></td>
</tr>
<tr>
<td>file</td>
<td>The file to download the private JSON key to</td>
</tr>
</tbody>
</table>

Details

It will download the existing roles, and append the role you add to it here. If you supply an accountId to gar_service_get_roles then it will return only those roles that accountId has.

Value

If it already exists, returns it via gar_service_get, else creates the service key.

See Also

Combine these functions to provision emails in one step with gar_service_provision
Other IAM functions: gar_service_provision()
Example

```r
# Not run:
# all roles
projectId <- gar_set_client(
  json = Sys.getenv("GAR_CLIENT_JSON"),
  scopes = "https://www.googleapis.com/auth/cloud-platform")

# roles for one accountID
gar_service_get_roles(
  projectId,
  accountId = "1080525199262@cloudbuild.gserviceaccount.com")
```

---

**gar_service_provision**

*Provision a service account*

**Description**

This uses all the `gar_service_create` functions to enable creating service account roles more easily.

**Usage**

```r
gar_service_provision(
  accountId,
```
gar_setup_auth_check

```r
roles,
    json = Sys.getenv("GAR_CLIENT_JSON"),
    file = paste0(accountId, "-auth-key.json"),
    email = Sys.getenv("GARGLE_EMAIL")
)
```

**Arguments**

- `accountId`: The service account ID
- `roles`: A character vector of roles to give the accountIds e.g. `roles/editor` - see list of roles here [https://cloud.google.com/iam/docs/understanding-roles#predefined_roles](https://cloud.google.com/iam/docs/understanding-roles#predefined_roles) or in your GCP console [https://console.cloud.google.com/iam-admin/roles/details/roles](https://console.cloud.google.com/iam-admin/roles/details/roles)
- `json`: The file location of an OAuth 2.0 client ID json file
- `file`: The file to download the private JSON key to
- `email`: An existing gargle cached email to authenticate with or TRUE to authenticate with the only email available.

**Details**

You will need the OAuth2.0 Client ID JSON from your GCP project via menu icon > APIs & Services > Credentials > Create Credentials > OAuth client ID

You need to authenticate with a user with permission `iam.serviceAccounts.create` for the project. Most often the user is an Owner/Editor

**See Also**

[https://cloud.google.com/iam/docs/creating-managing-service-accounts#iam-service-accounts-create-rest](https://cloud.google.com/iam/docs/creating-managing-service-accounts#iam-service-accounts-create-rest)

Other IAM functions: `gar_service_create()`

**Examples**

```r
## Not run:

gar_service_provision("my-service-account",
    c("roles/viewer", "roles/bigquery.jobUser"))

## End(Not run)
```

---

gar_setup_auth_check  Check service key works via environment argument

**Description**

Check service key works via environment argument
Usage

gar_setup_auth_check(
    env_arg = "GCE_AUTH_FILE",
    scope = "https://www.googleapis.com/auth/cloud-platform"
)

Arguments

    env_arg       The authentication environment argument
    scope         The scope of the GCP request

See Also

Other setup functions: gar_setup_auth_key(), gar_setup_clientid(), gar_setup_edit_renviron(),
gar_setup_env_check(), gar_setup_menu(), gar_setup_menu_do()

---

gar_setup_auth_key   Create a service account for googleCloudRunner

Description

This will use your Google OAuth2 user to create a suitable service account

Usage

gar_setup_auth_key(
    email = Sys.getenv("GARGLE_EMAIL"),
    file = "googleauthr-auth-key.json",
    session_user = NULL,
    client_json = "GAR_CLIENT_JSON",
    roles = NULL,
    default_key = "googleauthr"
)

Arguments

    email       What email to open OAuth2 with
    file        Where to save the authentication file
    session_user 1 for user level, 2 for project level, leave NULL to be prompted
    client_json The location of the env arg holding client json
    roles       Whether to assign roles to the service key
    default_key The default name of the service key

Value

TRUE if the file is ready to be setup, FALSE if need to stop
**gar_setup_clientid**

See Also

Other setup functions: `gar_setup_auth_check()`, `gar_setup_clientid()`, `gar_setup_edit_renviron()`, `gar_setup_env_check()`, `gar_setup_menu()`, `gar_setup_menu_do()`

---

### Description

Check for a client JSON

### Usage

```r
gar_setup_clientid(session_user = NULL, client_json = "GAR_CLIENT_JSON")
```

### Arguments

- **session_user**: 1 for user level, 2 for project level, leave NULL to be prompted
- **client_json**: The environment argument to be used for client_id/secret

### Value

TRUE is client_id is ready, FALSE if it is not

---

### See Also

Other setup functions: `gar_setup_auth_check()`, `gar_setup_auth_key()`, `gar_setup_edit_renviron()`, `gar_setup_env_check()`, `gar_setup_menu()`, `gar_setup_menu_do()`

---

### gar_setup_edit_renviron

*Setup wizard help - asking users to edit .Renviron*

---

### Description

Setup wizard help - asking users to edit .Renviron

### Usage

```r
gar_setup_edit_renviron(to_paste, session_user)
```

```r
gar_setup_check_session(session_user = NULL)
```
Arguments

- **to_paste**
  - The line to paste into .Renviron
- **session_user**
  - Whether it's a 1 = user level or 2 = project level .Renviron file
  - Intended to get user input from a menu, 1 indicating user level, 2 project level
  - `gar_setup_check_session` creates a menu for the user to choose which

See Also

Other setup functions: `gar_setup_auth_check()`, `gar_setup_auth_key()`, `gar_setup_clientid()`, `gar_setup_env_check()`, `gar_setup_menu()`, `gar_setup_menu_do()`

Examples

```r
## Not run:
choice <- gar_setup_check_session()
gar_setup_edit_renviron("ENV_ARG=blah", session_user = choice)
## End(Not run)
```

---

gar_setup_env_check  Setup wizard help - check if environment argument is set

Description

Setup wizard help - check if environment argument is set

Usage

```r
gar_setup_env_check(env_arg, set_to, edit_option = FALSE, session_user)
```

Arguments

- **env_arg**
  - The environment argument to check
- **set_to**
  - NULL or a string to set in .Renviron
- **edit_option**
  - Pass edit_option = FALSE to edit an existing environment arg
- **session_user**
  - 1 = user, 2 = project scope of .Renviron

Details

Pass edit_option = FALSE to edit an existing environment arg, TRUE will check if it exists, and will pass if its present.
**gar_setup_get_authenv**

Setup wizard helper - add authentication file to .Renviron

**Description**

Setup wizard helper - add authentication file to .Renviron

**Usage**

```r
gar_setup_get_authenv(env_arg = "GCE_AUTH_FILE", ...)
```

**Arguments**

- `env_arg` The environment argument to set
- `...` Other arguments passed to `gar_setup_auth_key`

**Value**

A string to paste into an .Renviron, or NULL

---

**gar_setup_menu** Setup wizard - introduction helper

**Description**

Salutation and initial menu

**Usage**

```r
gar_setup_menu(choices, package_name = "googleAuthR")
```

**Arguments**

- `choices` A character vector of the choices passed to `menu`
- `package_name` The package the setup menu is for
**gar_setup_menu_do**

**Value**

The number option from the menu

**See Also**

Other setup functions: `gar_setup_auth_check()`, `gar_setup_auth_key()`, `gar_setup_clientid()`, `gar_setup_edit_renviron()`, `gar_setup_env_check()`, `gar_setup_menu_do()`

---

**gar_setup_menu_do**  
*Setup wizard help - the functions that will execute on different menu options*

**Description**

Setup wizard help - the functions that will execute on different menu options

**Usage**

```r
gar_setup_menu_do(menu_option, trigger, do_function, stop = FALSE, ...)
```

**Arguments**

- `menu_option`: The menu option chosen from `menu` or `gar_setup_menu`
- `trigger`: What option will trigger the `do_function`
- `do_function`: The function in the same order as the menu options
- `stop`: Whether to stop and exit if the function comes back `FALSE`
- `...`: arguments passed to `do_function`

**Details**

The functions should come back with `TRUE` or `FALSE` depending on if the setting was successful.

**Value**

`FALSE` if setting was not set, `TRUE` if it was

**See Also**

Other setup functions: `gar_setup_auth_check()`, `gar_setup_auth_key()`, `gar_setup_clientid()`, `gar_setup_edit_renviron()`, `gar_setup_env_check()`, `gar_setup_menu()`
### Not run:

```r
op <- gar_setup_menu(c("Check all settings",
    "Configure authentication",
    "Configure env arg 1",
    "Configure env arg 2",
    "Configure something else" ),
    package_name = "googleAuthR")

choice <- gar_setup_check_session()

custom_env_check_f <- function(choice){
  r <- readline("project-id:")
  gar_setup_env_check("ARG2",
    set_to = r,
    edit_option = choice == 1, #allow editing of env arg
    session_user = choice)
  TRUE
}

gar_setup_menu_do(op, c(1,2), my_setup_auth_f, stop = TRUE)
gar_setup_menu_do(op, c(1,3), gar_setup_env_check,
    env_arg = "ARG1", set_to = "BLAH",
    edit_option = choice == 1, #allow editing of env arg
    session_user = choice)
gar_setup_menu_do(op, c(1,4), custom_env_check_f)
gar_setup_menu_do(op, c(1,4), my_setup_something_f)
```

## End(Not run)

---

**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**

```r
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 \texttt{options(googleAuthR.client_id)}, \texttt{options(googleAuthR.client_secret)} and \texttt{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 \texttt{options(googleAuthR.webapp.client_id)}, \texttt{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 \texttt{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:

\begin{verbatim}
  gar_set_client("google-client.json",
                 scopes = "http://www.googleapis.com/auth/webmasters")
  gar_auth_service("google-service-auth.json")
\end{verbatim}

## End(Not run)
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

```r
## 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"),
    prompt = c("consent", "select_account", "both", "none")
)

Arguments

req      a Rook request, do not set as this will be used by Shiny to generate URL
**gar_shiny_login_ui**

A login page for Shiny

---

**Description**

An alternative to the immediate login provided by default by `gar_shiny_ui`

**Usage**

```r
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()`, `gar_shiny_login_ui()`, `gar_shiny_ui()`, `silent_auth()`
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
```r
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 this gist by Joe Cheng, RStudio

See Also
Other pre-load shiny authentication: `gar_shiny_auth()`, `gar_shiny_auth_url()`, `gar_shiny_login_ui()`, `silent_auth()`

Examples
```r
## 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",
```
gar_token

## 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 \[gar_deauth()\], \texttt{gar_token()} returns \texttt{NULL}.

#### Usage

gar_token()

#### Value

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

#### See Also

Other low-level API functions: gar_has_token()
Examples

```r
## 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**

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

**Arguments**

- `detail_level` : How much info to show

---

**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.skip_token_checks = FALSE
• 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

Author(s)

Maintainer: Erik Grönroos <erik.gronroos@8-bit-sheep.com>

Authors:

• Mark Edmondson <m@sunholo.com> (ORCID)

Other contributors:

• Jennifer Bryan <jenny@stat.ubc.ca> [contributor]
• Johann deBoer <johannedeboer@gmail.com> [contributor]
• Neal Richardson <neal.p.richardson@gmail.com> [contributor]
• David Kulp <dkulp@cs.umass.edu> [contributor]
• Joe Cheng <joe@rstudio.com> [contributor]

See Also

Useful links:

• https://code.markedmondson.me/googleAuthR/
• Report bugs at https://github.com/MarkEdmondson1234/googleAuthR/issues

---

**googleSignIn**  
*Google Sign In [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**

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

input    shiny input (must contain g_id, g_name, g_email, g_image, g_signed_in)
output   shiny output (passed by shiny but not used)
session  shiny session

Details

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

Value

A reactive list with values $id, $name, $email, $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

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()`, `gar_shiny_auth_url()`, `gar_shiny_login_ui()`, `gar_shiny_ui()`

skip_if_no_env_auth

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
Index

* Google Discovery API functions
  gar_create_api_objects, 18
  gar_create_api_skeleton, 19
  gar_create_package, 19
  gar_discovery_api, 21
  gar_discovery_apis_list, 22

* IAM functions
  gar_service_create, 26
  gar_service_provision, 28

* auth functions
  gar_auth_configure, 9
  gar_deauth, 20

* authentication functions
  gar_attach_auto_auth, 6
  gar_auth, 7
  gar_auth_service, 10
  gar_auto_auth, 11
  gar_gce_auth, 22

* batch functions
  gar_batch, 12
  gar_batch_walk, 13

* cache functions
  gar_cache_get_loc, 16

* low-level API functions
  gar_has_token, 25
  gar_token, 41

* pre-load shiny authentication
  gar_shiny_auth, 37
  gar_shiny_auth_url, 38
  gar_shiny_login_ui, 39
  gar_shiny_ui, 40
  silent_auth, 45

* setup functions
  gar_setup_auth_check, 29
  gar_setup_auth_key, 30
  gar_setup_clientid, 31
  gar_setup_edit_reviron, 31
  gar_setup_env_check, 32
  gar_setup_menu, 33
  gar_setup_menu_do, 34
  .onAttach, 6, 7, 12

  add_headers, 3
  check, 20
  create_package, 20

  fromJSON, i, 9

  gar_api_generator, 3, 4, 12, 14
  gar_api_key (gar_auth_configure), 9
  gar_api_page, 4
  gar_attach_auto_auth, 6, 8, 11, 12, 23
  gar_auth, 7, 7, 11, 12, 23
  gar_auth_service, 7, 8, 10, 12, 23, 36
  gar_auto_auth, 7, 8, 11, 12, 23
  gar_batch, 12, 15
  gar_batch_walk, 12, 13
  gar_cache_empty (gar_cache_get_loc), 16
  gar_cache_get_loc, 16
  gar_cache_setup (gar_cache_get_loc), 16
  gar_check_existing_token, 17
  gar_create_api_objects, 18, 19, 20, 22
  gar_create_api_skeleton, 18, 19, 20, 22
  gar_create_package, 18, 19, 19, 22
  gar_deauth, 10, 20
  gar_debug_parsing, 21
  gar_discovery_api, 18–20, 21, 22
  gar_discovery_apis_list, 18–20, 22, 22
  gar_gce_auth, 7, 8, 11, 12, 22, 24, 25
  gar_gce_auth_default, 23
  gar_gce_auth_email, 23, 24
  gar_has_token, 25, 41
  gar_oauth_app (gar_auth_configure), 9
  gar_scope_config, 26
  gar_service_create, 26, 28, 29
  gar_service_get, 27
  gar_service_get (gar_service_create), 26

46
INDEX

gar_service_get_roles
  (gar_service_create), 26
use Github, 20

use_proxy, 3

use_github, 20

use_proxy, 3

gar_service_grant_roles
  (gar_service_create), 26

gar_service_key (gar_service_create), 26

gar_service_key_list
  (gar_service_create), 26

gar_service_list (gar_service_create), 26

gar_service_list (gar_service_create), 26

gar_service_provision, 27, 28

gar_set_client, 35

get_google_token, 7, 8, 11, 12, 23

googleAuthR-package (googleAuthR), 42

googleSignInUI, 43, 44

googleSignIn, 43, 44

googleAuthR, 42

googleSignInUI, 43, 44

memoise, 16

menu, 33, 34

shinyApp, 40

should_skip_token_checks, 7, 8, 11, 12, 23

silent_auth, 37, 39, 40, 45

skip_if_no_env_auth, 45

Sys.getenv, 11

tidy_eval, 18–20

Token2.0, 8, 12

token_exists, 7, 8, 11, 12, 23

token_fetch, 7