

# Package ‘googleAuthR’

April 11, 2023

**Type** Package

**Version** 2.0.1

**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** <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), 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), rstudioapi, shiny (>= 0.13.2), testthat, usethis (>= 1.6.0)

**License** MIT + file LICENSE

**VignetteBuilder** knitr

**RoxygenNote** 7.1.2

**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** 2023-04-11 10:10:02 UTC

**R topics documented:**

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

---

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 <a href="#">use_proxy</a> or <a href="#">add_headers</a> that will be added to the request.
simplifyVector	Passed to <a href="#">fromJSON</a> 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 the 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
## .Renviro 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"
)
```

**Arguments**

token	an actual token object or the path to a valid token stored as an <code>.rds</code> file
email	An existing gargle cached email to authenticate with or <code>TRUE</code> to authenticate with the only email available.
scopes	Scope of the request
app	app as specified by <a href="#">gar_auth_configure</a>
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\(\)](#), [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 <code>[httpr::oauth_app()]</code> .
path	JSON downloaded from Google Cloud Platform Console, containing a client id (aka key) and secret, in one of the forms supported for the <code>txt</code> argument of <code>jsonlite::fromJSON()</code> (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 `[httpr::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 (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 = "abcdefghijklmnopqrstuvwxy"
  )
  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\_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	<i>Perform auto authentication</i>
---------------	------------------------------------

---

**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 <a href="#">gar_auth</a> 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 <a href="#">gar_attach_auto_auth</a> . environment_var This is the name that will be called via <a href="#">Sys.getenv</a> 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(
  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**

<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 <a href="#">gar_api_generator</a>
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

<code>mcache</code>	A cache method from <a href="#">memoise</a> .
<code>invalid_func</code>	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 <a href="#">gar_discovery_api</a>
format	If TRUE will use <a href="#">tidy_eval</a> 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 <a href="#">gar_discovery_api</a>
format	If TRUE will use <a href="#">tidy_eval</a> 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 = FALSE,  
  github = FALSE,  
  format = TRUE,  
  overwrite = TRUE  
)
```

**Arguments**

api_json	json from <a href="#">gar_discovery_api</a>
directory	Where to build the package
rstudio	Passed to <a href="#">create_package</a> , creates RStudio project file
check	Perform a <a href="#">check</a> on the package once done
github	If TRUE will upload package to your github
format	If TRUE will use <a href="#">tidy_eval</a> 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>

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\\_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**

Download the discovery document for an API

**Usage**

```
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](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\(\)](#)

---

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 = "https://www.googleapis.com/auth/cloud-platform"
)
```

**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.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

[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	<i>Is there a token on hand?</i>
---------------	----------------------------------

---

**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_scope_config	<i>Create or add scopes to configuration</i>
------------------	--

---

**Description**

Helper for working with scopes

**Usage**

```
gar_scope_config(required_scopes)
```

**Arguments**

```
required_scopes
    character vector of scopes to add
```

---

gar_service_create	<i>Work with service accounts via the API</i>
--------------------	---

---

**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_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

accountId	The service accountId
projectId	The projectId containing the service account
serviceName	Name of service account
serviceDescription	Description of service account
accountIds	A vector of accountIds in the form accountId@projectId.iam.gserviceaccount.com
roles	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/role">https://console.cloud.google.com/iam-admin/roles/details/role</a>
type	The type of accountId to add role for - e.g. user:mark@me.com or serviceAccount:accountId@projectId
file	The file to download the private JSON key to

### 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](#)

<https://cloud.google.com/resource-manager/reference/rest/v1/projects/setIamPolicy>

<https://cloud.google.com/resource-manager/reference/rest/v1/projects/setIamPolicy>

<https://cloud.google.com/iam/docs/reference/rest/v1/projects.serviceAccounts.keys/create>

Other IAM functions: [gar\\_service\\_provision\(\)](#)

**Examples**

```
## Not run:

# all roles
projectId <- gar_set_client(
  json = Sys.getenv("GAR_CLIENT_JSON"),
  scopes = "https://www.googleapis.com/auth/cloud-platform")
gar_service_get_roles(projectId)

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

## End(Not run)
## Not run:
library(googleAuthR)
gar_set_client(scopes = "https://www.googleapis.com/auth/cloud-platform")
gar_auth()
gar_service_create("test12345678", "my-project")

gar_service_get("test12345678@my-project.iam.gserviceaccount.com",
  projectId = "my-project")

gar_service_grant_roles("test12345678@my-project.iam.gserviceaccount.com",
  role = "roles/editor",
  projectId = "my-project")

gar_service_key("test12345678", "my-project", "my-auth.json")

gar_service_list("my-project")

gar_service_key_list("test12345678", "my-project")

## End(Not run)
```

---

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**

```
gar_service_provision(
  accountId,
```

```

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

```

### Arguments

accountId	The service accountId
roles	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/editor">https://console.cloud.google.com/iam-admin/roles/details/roles/editor</a>
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-ro>

Other IAM functions: `gar_service_create()`

### Examples

```

## 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\\_renvirion\(\)](#), [gar\\_setup\\_env\\_check\(\)](#), [gar\\_setup\\_menu\\_do\(\)](#), [gar\\_setup\\_menu\(\)](#)

---

gar_setup_auth_key	<i>Create a service account for googleCloudRunner</i>
--------------------	---

---

**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

**See Also**

Other setup functions: [gar\\_setup\\_auth\\_check\(\)](#), [gar\\_setup\\_clientid\(\)](#), [gar\\_setup\\_edit\\_renviron\(\)](#), [gar\\_setup\\_env\\_check\(\)](#), [gar\\_setup\\_menu\\_do\(\)](#), [gar\\_setup\\_menu\(\)](#)

---

gar\_setup\_clientid      *Check for a client JSON*

---

**Description**

Check for a client JSON

**Usage**

```
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\\_do\(\)](#), [gar\\_setup\\_menu\(\)](#)

---

gar\_setup\_edit\_renviron  
*Setup wizard help - asking users to edit .Renviron*

---

**Description**

Setup wizard help - asking users to edit .Renviron

**Usage**

```
gar_setup_edit_renviron(to_paste, session_user)

gar_setup_check_session(session_user = NULL)
```

**Arguments**

to\_paste            The line to paste into .Renviron  
 session\_user        whether its 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\\_do\(\)](#), [gar\\_setup\\_menu\(\)](#)

**Examples**

```
## 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**

```
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.



**Value**

TRUE once changes made

**See Also**

Other setup functions: [gar\\_setup\\_auth\\_check\(\)](#), [gar\\_setup\\_auth\\_key\(\)](#), [gar\\_setup\\_clientid\(\)](#), [gar\\_setup\\_edit\\_renviron\(\)](#), [gar\\_setup\\_menu\\_do\(\)](#), [gar\\_setup\\_menu\(\)](#)

---

gar\_setup\_get\_authenv *Setup wizard helper - add authentication file to .Renviron*

---

**Description**

Setup wizard helper - add authentication file to .Renviron

**Usage**

```
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**

```
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

**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	<i>Setup wizard help - the functions that will execute on different menu options</i>
-------------------	--

---

**Description**

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

**Usage**

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

**Arguments**

menu_option	The menu option chosen from <a href="#">menu</a> or <a href="#">gar_setup_menu</a>
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\(\)](#)

**Examples**

```
## Not run:

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**

```
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"),
  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

state	URL state
client.id	client.id
client.secret	client.secret
scope	API scopes
access_type	whether to keep the token
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 <a href="#">gar_shiny_auth_url</a> 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

<code>ui</code>	A Shiny ui object
<code>login_ui</code>	A UI or HTML template that is seen before the main app and contains a login in link generated by <a href="#">gar_shiny_auth_url</a>

## 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\\_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	<i>Get current token summary</i>
----------------	----------------------------------

---

**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

---

googleAuthR	<i>googleAuthR: Easy Authentication with Google OAuth2 APIs</i>
-------------	---

---

**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

---

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 (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	<i>Google SignIn [UI Module]</i>
----------------	----------------------------------

---

**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	<i>Silent auth</i>
-------------	--------------------

---

**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

# 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, 44
- \* **setup functions**
  - gar\_setup\_auth\_check, 29
  - gar\_setup\_auth\_key, 30
  - gar\_setup\_clientid, 31
  - gar\_setup\_edit\_renviro, 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, 3
- 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\_configure, 8, 9, 20
- gar\_auth\_service, 7, 8, 10, 12, 23, 36
- gar\_auto\_auth, 7, 8, 11, 11, 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, 9, 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

gar\_service\_get\_roles  
    (gar\_service\_create), 26

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\_provision, 27, 28

gar\_set\_client, 35

gar\_setup\_auth\_check, 29, 31–34

gar\_setup\_auth\_key, 30, 30, 31–34

gar\_setup\_check\_session, 32

gar\_setup\_check\_session  
    (gar\_setup\_edit\_renviro), 31

gar\_setup\_clientid, 30, 31, 31, 32–34

gar\_setup\_edit\_renviro, 30, 31, 31, 33,  
    34

gar\_setup\_env\_check, 30–32, 32, 34

gar\_setup\_get\_authenv, 33

gar\_setup\_menu, 30–33, 33, 34

gar\_setup\_menu\_do, 30–34, 34

gar\_shiny\_auth, 37, 39, 40, 45

gar\_shiny\_auth\_url, 37, 38, 39, 40, 45

gar\_shiny\_login\_ui, 37, 39, 39, 40, 45

gar\_shiny\_ui, 37, 39, 40, 44, 45

gar\_token, 25, 41

gar\_token\_info, 42

get\_google\_token, 7, 8, 11, 12, 23

googleAuthR, 42

googleSignIn, 43, 44

googleSignInUI, 43, 44

jsonlite::fromJSON(), 9

memoise, 16

menu, 33, 34

shinyApp, 40

silent\_auth, 37, 39, 40, 44

skip\_if\_no\_env\_auth, 45

Sys.getenv, 11

tidy\_eval, 18–20

Token2.0, 8, 11

token\_exists, 7, 8, 11, 12, 23

token\_fetch, 7

use\_github, 20

use\_proxy, 3