Package ‘gargle’

October 4, 2019

Title Utilities for Working with Google APIs
Version 0.4.0

Description Provides utilities for working with Google APIs
<https://developers.google.com/apis-explorer>. This includes
functions and classes for handling common credential types and for
preparing, executing, and processing HTTP requests.

License MIT + file LICENSE


BugReports https://github.com/r-lib/gargle/issues

Depends R (>= 3.2)

Imports fs (>= 1.3.1), glue (>= 1.3.0), htrr (>= 1.4.0), jsonlite,
rlang (>= 0.4.0), stats, withr

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

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

NeedsCompilation no

Author Jennifer Bryan [aut, cre] (<https://orcid.org/0000-0002-6983-2759>),
Craig Citro [aut],
Hadley Wickham [aut] (<https://orcid.org/0000-0003-4757-117X>),
Google Inc [cph],
RStudio [cph, fnd]

Maintainer Jennifer Bryan <jenny@rstudio.com>

Repository CRAN

Date/Publication 2019-10-04 05:40:05 UTC
R topics documented:

<table>
<thead>
<tr>
<th>R topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthState-class</td>
<td>2</td>
</tr>
<tr>
<td>credentials_app_default</td>
<td>3</td>
</tr>
<tr>
<td>credentials_byo_oauth2</td>
<td>4</td>
</tr>
<tr>
<td>credentials_gce</td>
<td>5</td>
</tr>
<tr>
<td>credentials_service_account</td>
<td>6</td>
</tr>
<tr>
<td>credentials_user_oauth2</td>
<td>7</td>
</tr>
<tr>
<td>cred_funs</td>
<td>9</td>
</tr>
<tr>
<td>gargle2.0_token</td>
<td>10</td>
</tr>
<tr>
<td>gargle_app</td>
<td>11</td>
</tr>
<tr>
<td>gargle_oauth_sitrep</td>
<td>12</td>
</tr>
<tr>
<td>gargle_options</td>
<td>13</td>
</tr>
<tr>
<td>GceToken</td>
<td>14</td>
</tr>
<tr>
<td>init_AuthState</td>
<td>14</td>
</tr>
<tr>
<td>oauth_app_from_json</td>
<td>15</td>
</tr>
<tr>
<td>request_develop</td>
<td>16</td>
</tr>
<tr>
<td>request_make</td>
<td>19</td>
</tr>
<tr>
<td>response_process</td>
<td>20</td>
</tr>
<tr>
<td>token-info</td>
<td>22</td>
</tr>
<tr>
<td>token.fetch</td>
<td>23</td>
</tr>
</tbody>
</table>

**Index**

<table>
<thead>
<tr>
<th>R topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthState-class</td>
<td>24</td>
</tr>
</tbody>
</table>

**Description**

An `AuthState` object manages an authorization state, typically on behalf of a client package that makes requests to a Google API.

The [How to use gargle for auth in a client package](#) vignette describes a design for wrapper packages that relies on an `AuthState` object. This state can then be incorporated into the package’s requests for tokens and can control the inclusion of tokens in requests to the target API.

- **api_key** is the simplest way to associate a request with a specific Google Cloud Platform project. A few calls to certain APIs, e.g. reading a public Sheet, can succeed with an API key, but this is the exception.
- **app** is an OAuth app associated with a specific Google Cloud Platform project. This is used in the OAuth flow, in which an authenticated user authorizes the app to access or manipulate data on their behalf.
- **auth_active** reflects whether outgoing requests will be authorized by an authenticated user or are unauthorized requests for public resources. These two states correspond to sending a request with a token versus an API key, respectively.
- **cred** is where the current token is cached within a session, once one has been fetched. It is generally assumed to be an instance of `httr::TokenServiceAccount` or `httr::Token2.0` (or a subclass thereof), probably obtained via `token_fetch()` (or one of its constituent credential fetching functions).
An AuthState should be created through the constructor function `init_AuthState()`.

Usage

AuthState

Format

An R6 class object.

Description

Loads credentials from a file identified via a search strategy known as Application Default Credentials (ADC). The hope is to make auth "just work" for someone working on Google-provided infrastructure or who has used Google tooling to get started. A sequence of paths is consulted, which we describe here, with some abuse of notation. ALL_CAPS represents the value of an environment variable and %||% is used in the spirit of a null coalescing operator.

```
GOOGLE_APPLICATION_CREDENTIALS
CLOUDSDK_CONFIG/application_default_credentials.json
# on Windows:
(APPDATA %||% SystemDrive %||% C:\)\gcloud\application_default_credentials.json
# on not-Windows:
~/.config/gcloud/application_default_credentials.json
```

If the above search successfully identifies a JSON file, it is parsed and ingested either as a service account token or a user OAuth2 credential.

Usage

```
credentials_app_default(scopes = NULL, ...)
```

Arguments

- **scopes**: A character vector of scopes to request. Pick from those listed at [https://developers.google.com/identity/protocols/googlescopes](https://developers.google.com/identity/protocols/googlescopes). For certain token flows, the "https://www.googleapis.com/auth/userinfo.email" scope is unconditionally included. This grants permission to retrieve the email address associated with a token; gargle uses this to index cached OAuth tokens. This grants no permission to view or send email. It is considered a low value scope and does not appear on the consent screen.

- **...**: Additional arguments passed to all credential functions.
Value

An \texttt{httr::TokenServiceAccount} or an \texttt{httr::Token2.0} or NULL.

See Also

\url{https://cloud.google.com/docs/authentication/production#providing_credentials_to_your_application}
\url{https://cloud.google.com/sdk/docs/}

Other credential functions: \texttt{credentials_byo_oauth2, credentials_gce, credentials_service_account, credentials_user_oauth2, token_fetch}

Examples

\begin{verbatim}
## Not run:
credentials_app_default()

## End(Not run)
\end{verbatim}

credentials\_byo\_oauth2

\emph{Load a user-provided token}

Description

This function does very little when called directly with a token:

- If input has class \texttt{request}, i.e. it is a token that has been prepared with \texttt{httr::config()}, the auth\_token component is extracted. For example, such input could be produced by \texttt{googledrive::drive\_token()} or \texttt{bigrquery::bq\_token()}.
- Checks that the input appears to be a Google OAuth token, based on the embedded oauth\_endpoint.
- Refreshes the token, if it's refreshable.
- Returns its input.

There is no point providing scopes. They are ignored because the scopes associated with the token have already been baked in to the token itself and gargle does not support incremental authorization. The main point of \texttt{credentials\_byo\_oauth2()} is to allow \texttt{token\_fetch()} (and packages that wrap it) to accomodate a "bring your own token" workflow.

This also makes it possible to obtain a token with one package and then register it for use with another package. For example, the default scope requested by googledrive is also sufficient for operations available in googlesheets4. You could use a shared token like so:

\begin{verbatim}
library(googledrive)
library(googlesheets4)
drive_auth(email = "jane_doe@example.com")
sheets_auth(token = drive_token())
# work with both packages freely now
\end{verbatim}
**credentials_gce**

Usage

```
credentials_byo_oauth2(scopes = NULL, token, ...)
```

Arguments

- **scopes**: A character vector of scopes to request. Pick from those listed at https://developers.google.com/identity/protocols/googlescopes. For certain token flows, the "https://www.googleapis.com/auth/userinfo.email" scope is unconditionally included. This grants permission to retrieve the email address associated with a token; gargle uses this to index cached OAuth tokens. This grants no permission to view or send email. It is considered a low value scope and does not appear on the consent screen.

- **token**: A token with class `Token2.0` or an object of `httr`'s class `request`, i.e. a token that has been prepared with `http::config()` and has a `Token2.0` in the `auth_token` component.

- **...**: Additional arguments passed to all credential functions.

Value

An `Token2.0`.

See Also

Other credential functions: `credentials_app_default`, `credentials_gce`, `credentials_service_account`, `credentials_user_oauth2`, `token_fetch`

Examples

```
## Not run:
# assume `my_token` is a Token2.0 object returned by a function such as
# http::oauth2.0_token() or gargle::gargle2.0_token()
credentials_byo_oauth2(token = my_token)
```

Description

Uses the metadata service available on GCE VMs to fetch an access token.

Usage

```
credentials_gce(scopes = "https://www.googleapis.com/auth/cloud-platform",
               service_account = "default", ...)```
Arguments


For certain token flows, the "https://www.googleapis.com/auth/userinfo.email" scope is unconditionally included. This grants permission to retrieve the email address associated with a token; gargle uses this to index cached OAuth tokens. This grants no permission to view or send email. It is considered a low value scope and does not appear on the consent screen.

service_account Name of the GCE service account to use.

... Additional arguments passed to all credential functions.

Value

A GceToken() or NULL.

See Also

https://cloud.google.com/compute/docs/storing-retrieving-metadata

Other credential functions: credentials_app_default, credentials_byo_oauth2, credentials_service_account, credentials_user_oauth2, token_fetch

Examples

## Not run:
credentials_gce()

## End(Not run)

---

credentials_service_account

Load a service account token

Description

Load a service account token

Usage

credentials_service_account(scopes = NULL, path = "", ...)
credentials_user_oauth2

Arguments

scopes A character vector of scopes to request. Pick from those listed at https://developers.google.com/identity/protocols/googlescopes. For certain token flows, the "https://www.googleapis.com/auth/userinfo.email" scope is unconditionally included. This grants permission to retrieve the email address associated with a token; gargle uses this to index cached OAuth tokens. This grants no permission to view or send email. It is considered a low value scope and does not appear on the consent screen.

path JSON identifying the service account, in one of the forms supported for the txt argument of jsonlite::fromJSON() (typically, a file path or JSON string).

... Additional arguments passed to all credential functions.

Value

An http::TokenServiceAccount or NULL.

See Also

Other credential functions: credentials_app_default, credentials_byo_oauth2, credentials_gce, credentials_user_oauth2, token_fetch

Examples

## Not run:
token <- credentials_service_account(
  scopes = "https://www.googleapis.com/auth/userinfo.email",
  path = "/path/to/your/service-account.json"
)
## End(Not run)

credentials_user_oauth2

*Get an OAuth token for a user*

Description

Consults the token cache for a suitable OAuth token and, if unsuccessful, gets a token via the browser flow. A cached token is suitable if it’s compatible with the user’s request in this sense:

- OAuth app must be same.
- Scopes must be same.
- Email, if provided, must be same.

gargle is very conservative about using OAuth tokens discovered in the user’s cache and will generally seek interactive confirmation. Therefore, in a non-interactive setting, it’s important to explicitly specify the "email" of the target account or to explicitly authorize automatic discovery. See gargle2.0_token(), which this function wraps, for more. Non-interactive use also suggests it might be time to use a service account token.
credentials_user_oauth2

Usage

```r
credentials_user_oauth2(scopes = NULL, app = gargle_app(),
                      package = "gargle", ...)
```

Arguments

  For certain token flows, the "https://www.googleapis.com/auth/userinfo.email" scope is unconditionally included. This grants permission to retrieve the email address associated with a token; gargle uses this to index cached OAuth tokens. This grants no permission to view or send email. It is considered a low value scope and does not appear on the consent screen.

- **app**: An OAuth consumer application, created by `httr::oauth_app()`.

- **package**: Name of the package requesting a token. Used in messages.

- **...**: Arguments passed on to `gargle2.0_token`

  - **email**: Optional. Allows user to target a specific Google identity. If specified, this is used for token lookup, i.e. to determine if a suitable token is already available in the cache. If no such token is found, email is used to pre-select the targeted Google identity in the OAuth chooser. Note, however, that the email associated with a token when it's cached is always determined from the token itself, never from this argument. Use NA or FALSE to match nothing and force the OAuth dance in the browser. Use TRUE to allow email auto-discovery, if exactly one matching token is found in the cache. Defaults to the option named "gargle_oauth_email", retrieved by `gargle::gargle_oauth_email()`.

  - **use_oob**: Whether to prefer "out of band" authentication. Defaults to the option named "gargle_oob_default", retrieved via `gargle::gargle_oob_default()`.

  - **cache**: Specifies the OAuth token cache. Defaults to the option named "gargle_oauth_cache", retrieved via `gargle::gargle_oauth_cache()`.

  - **user_params**: Named list holding endpoint specific parameters to pass to the server when posting the request for obtaining or refreshing the access token.

  - **type**: content type used to override incorrect server response

  - **credentials**: Advanced use only: allows you to completely customise token generation.

Value

A Gargle2.0 token.

See Also

Other credential functions: `credentials_app_default`, `credentials_byo_oauth2`, `credentials_gce`, `credentials_service_account`, `token_fetch`
cred_funs

Examples

## Not run:
## Drive scope, built-in gargle demo app
scopes <- "https://www.googleapis.com/auth/drive"
credentials_user_oauth2(scopes, app = gargle_app())

## bring your own app
app <- httr::oauth_app(
  appname = "my_awesome_app",
  key = "keykeykeykeykeykey",
  secret = "secretsecretsecret"
)
credentials_user_oauth2(scopes, app)

## End(Not run)

---

cred_funs  | Credential function registry

Description

Functions to query or manipulate the registry of credential functions consulted by `token_fetch()`.

Usage

cred_funs_list()  
cred_funs_add(...)  
cred_funs_set(ls)  
cred_funs_clear()  
cred_funs_set_default()

Arguments

...  One or more functions with the right signature: its first argument is named scopes, and it includes ... as an argument.
ls  A list of credential functions.

Value

A list of credential functions or NULL.
Functions

• cred_funs_list: Get the list of registered credential functions.

• cred_funs_add: Register one or more new credential fetching functions. Function(s) are added to the front of the list. So:

  * "First registered, last tried."
  * "Last registered, first tried."

• cred_funs_set: Register a list of credential fetching functions.

• cred_funs_clear: Clear the credential function registry.

• cred_funs_set_default: Reset the registry to the gargle default.

See Also

token_fetch(), which is where the registry is actually used.

Examples

names(cred_funs_list())
creds_one <- function(scopes, ...) {}
cred_funs_add(creds_one)
cred_funs_add(one = creds_one)
cred_funs_add(one = creds_one, two = creds_one)
cred_funs_add(one = creds_one, creds_one)

# undo all of the above and return to default
cred_funs_set_default()

gargle2.0_token Generate a gargle token

Description

Constructor function for objects of class Gargle2.0.

Usage

gargle2.0_token(email = gargle_oauth_email(), app = gargle_app(),
    package = "gargle", scope = NULL, user_params = NULL,
    type = NULL, use_oob = gargle_oob_default(), credentials = NULL,
    cache = if (is.null( credentials)) gargle_oauth_cache() else FALSE, ...)

Arguments

Arguments

email
Optional. Allows user to target a specific Google identity. If specified, this
is used for token lookup, i.e. to determine if a suitable token is already avail-
able in the cache. If no such token is found, email is used to pre-select
the targeted Google identity in the OAuth chooser. Note, however, that the email
associated with a token when it’s cached is always determined from the token
itself, never from this argument. Use NA or FALSE to match nothing and force
the OAuth dance in the browser. Use TRUE to allow email auto-discovery, if ex-
actly one matching token is found in the cache. Defaults to the option named
"gargle_oauth_email", retrieved by gargle::gargle_oauth_email().

app
An OAuth consumer application, created by httr::oauth_app().

package
Name of the package requesting a token. Used in messages.

scope
A character vector of scopes to request.

user_params
Named list holding endpoint specific parameters to pass to the server when post-
ing the request for obtaining or refreshing the access token.

type
content type used to override incorrect server response

use_oob
Whether to prefer “out of band” authentication. Defaults to the option named
"gargle_oob_default", retrieved via gargle::gargle_oob_default().

credentials
Advanced use only: allows you to completely customise token generation.

cache
Specifies the OAuth token cache. Defaults to the option named "gargle_oauth_cache",
retrieved via gargle::gargle_oauth_cache().

... Absorbs arguments intended for use by other credential functions. Not used.

Value
An object of class Gargle2.0, either new or loaded from the cache.

Examples

## Not run:
gargle2.0_token()

## End(Not run)

---

gargle_app OAuth app for demonstration purposes

Description
Invisibly returns an OAuth app that can be used to test drive gargle before obtaining your own client
ID and secret. This OAuth app may be deleted or rotated at any time. There are no guarantees
about which APIs are enabled. DO NOT USE THIS IN A PACKAGE or for anything other than
interactive, small-scale experimentation.

You can get your own OAuth app (client ID and secret), without these limitations. See the How to
get your own API credentials vignette for more details.
Usage

gargle_app()

Value

An OAuth consumer application, produced by `httr::oauth_app()`, invisibly.

Examples

```r
## Not run:
gargle_app()
## End(Not run)
```

gargle_oauth_sitrep

OAuth token situation report

Description

Get a human-oriented overview of the existing gargle OAuth tokens:

- Filepath of the current cache
- Number of tokens found there
- Compact summary of the associated
  - Email = Google identity
  - OAuth app (actually, just its nickname)
  - Scopes
  - Hash (actually, just the first 7 characters) Mostly useful for the development of gargle and client packages.

Usage

gargle_oauth_sitrep(cache = NULL)

Arguments

- `cache` Specifies the OAuth token cache. Defaults to the option named "gargle_oauth_cache", retrieved via `gargle::gargle_oauth_cache()`.

Value

A data frame with one row per cached token, invisibly.

Examples

```r
gargle_oauth_sitrep()
```
gargle_options

Options consulted by gargle

Description

Wrapper functions around options consulted by gargle, which provide:

- A place to hang documentation.
- The mechanism for setting a default.

If the built-in defaults don't suit you, set one or more of these options. Typically, this is done in the .Rprofile startup file, with code along these lines:

```r
options(
  gargle_oauth_email = "jane@example.com",
  gargle_oauth_cache = "/path/to/folder/that/does/not/sync/to/cloud"
)
```

Usage

- `gargle_oauth_email()`
- `gargle_oob_default()`
- `gargle_oauth_cache()`
- `gargle_quiet()`

**gargle_oauth_email**

`gargle_oauth_email()` returns the option named "gargle_oauth_email", which is undefined by default. If set, this option should be one of:

- An actual email address corresponding to your preferred Google identity. Example: janedoe@gmail.com.
- TRUE to allow email and OAuth token auto-discovery, if exactly one suitable token is found in the cache.
- FALSE or NA to force the OAuth dance in the browser.

**gargle_oob_default**

`gargle_oob_default()` returns the option named "gargle_oob_default", falls back to the option named "httr_oob_default", and eventually defaults to FALSE. This controls whether to prefer "out of band" authentication. This is ultimately passed to `httr::init_oauth2.0()` as use_oob. If FALSE (and httpuv is installed), a local webserver is used for the OAuth dance. Otherwise, user gets a URL and prompt for a validation code.

Read more about "out of band" authentication in the vignette Auth when using R in the browser.
gargle_oauth_cache

gargle_oauth_cache() returns the option named "gargle_oauth_cache", defaulting to NA. If defined, the option must be set to a logical value or a string. TRUE means to cache using the default user-level cache file, ~/.R/gargle/gargle-oauth, FALSE means don’t cache, and NA means to guess using some sensible heuristics.

gargle_quiet

gargle_quiet() returns the option named "gargle_quiet", which defaults to TRUE. Set this option to FALSE to see more info about gargle’s activities, which can be helpful for troubleshooting.

Examples

gargle_oauth_email()
gargle_oob_default()
gargle_oauth_cache()
gargle_quiet()

GceToken

Token for use on Google Compute Engine instances

Description

This class uses the metadata service available on GCE VMs to fetch access tokens. Not intended for direct use. See credentials_gce() instead.

Usage

GceToken

Format

An object of class R6ClassGenerator of length 24.

init_AuthState

Create an AuthState

Description

Constructor function for objects of class AuthState.

Usage

init_AuthState(package = NA_character_, app = NULL, api_key = NULL, auth_active = TRUE, cred = NULL)
oauth_app_from_json

Arguments

package  
Package name, an optional string. The associated package will generally by implied by the namespace within which the AuthState is defined. But it’s possible to record the package name explicitly and seems like a good practice.

app  
Optional. An OAuth consumer application, as produced by `httr::oauth_app()`.

api_key  
Optional. API key (a string). Some APIs accept unauthorized, "token-free" requests for public resources, but only if the request includes an API key.

auth_active  
Logical. TRUE means requests should include a token (and probably not an API key). FALSE means requests should include an API key (and probably not a token).

cred  
Credentials. Typically populated indirectly via `token_fetch()`.

Value

An object of class AuthState.

Examples

```r
my_app <- httr::oauth_app(
    appname = "my_package",
    key = "keykeykeykeykeykey",
    secret = "secretsecretsecret"
)

init_AuthState(
    package = "my_package",
    app = my_app,
    api_key = "api_key_api_key_api_key",
)
```

oauth_app_from_json  
Create an OAuth app from JSON

Description

Essentially a wrapper around `httr::oauth_app()` that extracts the necessary info from JSON obtained from Google Cloud Platform Console. If no appname is given, the "project_id" from the JSON is used.

Usage

```r
oauth_app_from_json(path, appname = NULL)
```
## request_develop

**Build a Google API request**

### Description

Intended primarily for internal use in client packages that provide high-level wrappers for users. The vignette Request helper functions describes how one might use these functions inside a wrapper package.

### Usage

```r
request_develop(endpoint, params = list(),
                 base_url = "https://www.googleapis.com")

request_build(method = "GET", path = ", path = "path to the JSON you downloaded from gcp/console.json",
               params = list(),
               body = list(), token = NULL, key = NULL,
               base_url = "https://www.googleapis.com")
```

### Arguments

- **endpoint**: List of information about the target endpoint or, in Google's vocabulary, the target "method". Presumably prepared from the Discovery Document for the target API.
- **params**: Named list. Values destined for URL substitution, the query, or, for request_develop(), only, the body. For request_build(), body parameters must be passed via the body argument.
- **base_url**: Character.
- **method**: Character. An HTTP verb, such as GET or POST.

### Examples

```r
## Not run:
oauth_app(
  path = "/path/to/the/JSON/you/downloaded/from/gcp/console.json"
)

## End(Not run)
```
**request_develop**

- **path**: Character. Path to the resource, not including the API’s base_url. Examples: drive/v3/about or drive/v3/files/{fileId}. The path can be a template, i.e. it can include variables inside curly brackets, such as {fileId} in the example. Such variables are substituted by `request_build()`, using named parameters found in `params`.

- **body**: List. Values to send in the API request body.

- **token**: Token, ready for inclusion in a request, i.e. prepared with `httr::config()`.

- **key**: API key. Needed for requests that don’t contain a token. For more, see Google’s document [Credentials, access, security, and identity](https://developers.google.com/). A key can be passed as a named component of `params`, but note that the formal argument `key` will clobber it, if non-NULL.

### Value

- `request_develop()`: list() with components `method`, `path`, `params`, `body`, and `base_url`.
- `request_build()`: list() with components `method`, `path` (post-substitution), `query` (the input `params` not used in URL substitution), `body`, `token`, `url` (the full URL, post-substitution, including the query).

### request_develop()

Combines user input (`params`) with information about an API endpoint. `endpoint` should contain these components:

- **path**: See documentation for argument.
- **method**: See documentation for argument.
- **parameters**: Compared with `params` supplied by user. An error is thrown if user-supplied `params` aren’t named in `endpoint$parameters` or if user fails to supply all required parameters. In the return value, body parameters are separated from those destined for path substitution or the query.

The return value is typically used as input to `request_build()`.

### request_build()

Builds a request, in a purely mechanical sense. This function does nothing specific to any particular Google API or endpoint.

- Use with the output of `request_develop()` or with hand-crafted input.
- `params` are used for variable substitution in `path`. Leftover `params` that are not bound by the `path` template automatically become HTTP query parameters.
- Adds an API key to the query iff `token = NULL` and removes the API key otherwise. Client packages should generally pass their own API key in, but note that `gargle_api_key()` is available for small-scale experimentation.

See `googledrive::generate_request()` for an example of usage in a client package. googledrive has an internal list of selected endpoints, derived from the [Drive API Discovery Document](https://developers.google.com/), exposed via `googledrive::drive_endpoints()`. An element from such a list is the expected input for
endpoint. googledrive::generate_request() is a wrapper around request_develop() and request_build() that inserts a googledrive-managed API key and some logic about Team Drives. All user-facing functions use googledrive::generate_request() under the hood.

**See Also**

Other requests and responses: request_make, response_process

**Examples**

```r
## Not run:
## Example with a prepared endpoint
ept <- googledrive::drive_endpoints("drive.files.update")[[1]]
req <- request_develop(
  ept,
  params = list(
    fileId = "abc",
    addParents = "123",
    description = "Exciting File"
  )
)
req <- request_build(
  method = req$method,
  path = req$path,
  params = req$params,
  body = req$body,
  token = "PRETEND_I_AM_A_TOKEN"
)
req

## Example with no previous knowledge of the endpoint
## List a file's comments
## https://developers.google.com/drive/v3/reference/comments/list
req <- request_build(
  method = "GET",
  path = "drive/v3/files/{fileId}/comments",
  params = list(
    fileId = "your-file-id-goes-here",
    fields = "*"
  ),
  token = "PRETEND_I_AM_A_TOKEN"
)
req

# Example with no previous knowledge of the endpoint and no token
# use an API key for which the Places API is enabled!
API_KEY <- "1234567890"

# get restaurants close to a location in Vancouver, BC
req <- request_build(

```
method = "GET",
path = "maps/api/place/nearbysearch/json",
params = list(
    location = "49.268682,-123.167117",
    radius = 100,
    type = "restaurant"
),
key = API_KEY,
base_url = "https://maps.googleapis.com"
)
resp <- request_make(req)
out <- response_process(resp)
vapply(out$results, function(x) x$name, character(1))
## End(Not run)

---

**request_make**  
*Make a Google API request*

**Description**

Intended primarily for internal use in client packages that provide high-level wrappers for users. `request_make()` does very little: calls an HTTP method, only adding a user agent. Typically the input is created with `request_build()` and the output is processed with `response_process()`.

**Usage**

```r
request_make(x, ..., user_agent = gargle_user_agent())
```

**Arguments**

- **x**: List. Holds the components for an HTTP request, presumably created with `request_develop()` or `request_build()`. Must contain a method and url. If present, body and token are used.
- **...**: Optional arguments passed through to the HTTP method.
- **user_agent**: A user agent string, prepared by `httr::user_agent()`. When in doubt, a client package should have an internal function that extends `gargle_user_agent()` by prepending its return value with the client package’s name and version.

**Value**

Object of class `response` from `httr`.

**See Also**

Other requests and responses: `request_develop`, `response_process`
## Not run:

```r
req <- gargle::request_build(
  method = "GET",
  path = "path/to/the/resource",
  token = "PRETEND_I_AM_TOKEN"
)
gargle::request_make(req)
```

## End(Not run)

---

**response_process**  
*Process a Google API response*

### Description

`response_process()` is intended primarily for internal use in client packages that provide high-level wrappers for users. Typically applied as the final step in this sequence of calls:

- Request prepared with `request_build()`.
- Request made with `request_make()`.
- Response processed with `response_process()`.

All that’s needed for a successful request is to parse the JSON extracted via `httr::content()`. Therefore, the main point of `response_process()` is to handle less happy outcomes:

- Status codes in the 400s (client error) and 500s (server error). The structure of the error payload varies across Google APIs and we try to create a useful message for all variants we know about.
- Non-JSON content type, such as HTML.
- Status code in the 100s (information) or 300s (redirection). These are unexpected.

### Usage

```r
response_process(resp, error_message = gargle_error_message)
```

```r
response_as_json(resp)
```

```r
gargle_error_message(resp)
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resp</code></td>
<td>Object of class response from <code>httr</code>.</td>
</tr>
<tr>
<td><code>error_message</code></td>
<td>Function that produces an informative error message from the primary input, <code>resp</code>. It must return a character vector.</td>
</tr>
</tbody>
</table>
Details

If `process_response()` results in an error, a redacted version of the `resp` input is returned in the condition (auth tokens are removed). Use functions such as `rlang::last_error()` or `rlang::catch_cnd()` to capture the condition and do a more detailed forensic examination.

The `response_as_json()` helper is exported only as an aid to maintainers who wish to use their own `error_message` function, instead of gargle’s built-in `gargle_error_message()`. When implementing a custom `error_message` function, call `response_as_json()` immediately on the input in order to inherit gargle’s handling of non-JSON input.

Value

The content of the request, as a list. An HTTP status code of 204 (No content) is a special case returning TRUE.

See Also

Other requests and responses: `request_develop`, `request_make`

Examples

```r
## Not run:
# get an OAuth2 token with 'userinfo.email' scope
token <- token_fetch(scopes = "https://www.googleapis.com/auth/userinfo.email")

# see the email associated with this token
req <- gargle::request_build(
  method = "GET",
  path = "v1/userinfo",
  token = token,
  base_url = "https://openidconnect.googleapis.com"
)
resp <- gargle::request_make(req)
response_process(resp)

# make a bad request (this token has incorrect scope)
req <- gargle::request_build(
  method = "GET",
  path = "fitness/v1/users/{userId}/dataSources",
  token = token,
  params = list(userId = 12345)
)
resp <- gargle::request_make(req)
response_process(resp)

## End(Not run)
```
Get info from a token

Description

These functions send the token to Google endpoints that return info about a token or a user.

Usage

- `token_userinfo(token)`
- `token_email(token)`
- `token_tokeninfo(token)`

Arguments

- `token` A token with class `Token2.0` or an object of `httr`'s class `request`, i.e. a token that has been prepared with `httr::config()` and has a `Token2.0` in the `auth_token` component.

Details

It's hard to say exactly what info will be returned by the "userinfo" endpoint targeted by `token_userinfo()`. It depends on the token's scopes. OAuth2 tokens obtained via the gargle package include the `https://www.googleapis.com/auth/userinfo.email` scope, which guarantees we can learn the email associated with the token. If the token has the `https://www.googleapis.com/auth/userinfo.profile` scope, there will be even more information available. But for a token with unknown or arbitrary scopes, we can't make any promises about what information will be returned.

Value

A list containing:

- `token_userinfo()`: user info
- `token_email()`: user's email (obtained from a call to `token_userinfo()`)
- `token_tokeninfo()`: token info

Examples

```r
# Not run:
# with service account token
t <- token_fetch(
  scopes = "https://www.googleapis.com/auth/drive",
  path = "path/to/service/account/token/blah-blah-blah.json"
)
# or with an OAuth token
t <- token_fetch(
```

token_fetch

```r
cscopes = "https://www.googleapis.com/auth/drive",
    email  = "janedoe@example.com"
)
token_userinfo(t)
token_email(t)
tokens_tokeninfo(t)

## End(Not run)
```

---

**token_fetch**

*Fetch a token for the given scopes*

**Description**

This is a rather magical function that calls a series of concrete credential-finding functions, each wrapped in a `tryCatch()`. `token_fetch()` keeps trying until it succeeds or there are no more functions to try. Use `cred_funs_list()` to see the current registry, in order. See the vignette *How gargle gets tokens* for a full description of `token_fetch()`.

**Usage**

```r
token_fetch(scopes = NULL, ...)
```

**Arguments**

- **scopes**
  

  For certain token flows, the "https://www.googleapis.com/auth/userinfo.email" scope is unconditionally included. This grants permission to retrieve the email address associated with a token; gargle uses this to index cached OAuth tokens. This grants no permission to view or send email. It is considered a low value scope and does not appear on the consent screen.

- `...`
  
  Additional arguments passed to all credential functions.

**Value**

An `httr::Token` or `NULL`.

**See Also**

Other credential functions: `credentials_app_default`, `credentials_byo_oauth2`, `credentials_gce`, `credentials_service_account`, `credentials_user_oauth2`

**Examples**

```r
## Not run:
token_fetch(scopes = "https://www.googleapis.com/auth/userinfo.email")

## End(Not run)
```
Index

*Topic datasets
  AuthState-class, 2
  GceToken, 14

AuthState, 14, 15
AuthState (AuthState-class), 2
AuthState-class, 2

cred_funs, 9
cred_funs_add (cred_funs), 9
cred_funs_clear (cred_funs), 9
cred_funs_list (cred_funs), 9
cred_funs_list(), 23
cred_funs_set (cred_funs), 9
cred_funs_set_default (cred_funs), 9
credentials_app_default, 3, 5–8, 23
credentials_byo_oauth2, 4, 4, 6–8, 23
credentials_gce, 4, 5, 7, 8, 23
credentials_gce(), 14
credentials_service_account, 4–6, 6, 8, 23
credentials_user_oauth2, 4–7, 7, 23

Gargle2.0, 8, 10, 11
gargle2.0_token, 10
gargle2.0_token(), 7
gargle::gargle_oauth_cache(), 8, 11, 12

gargle::gargle_oauth_email(), 8, 11

gargle::gargle_oob_default(), 8, 11

gargle_api_key(), 17
gargle_app, 11
gargle_error_message
  (response_process), 20

gargle_oauth_cache (gargle_options), 13
gargle_oauth_email (gargle_options), 13
gargle_oauth_sitrep, 12
gargle_oauth_default (gargle_options), 13
gargle_options, 13
gargle_quiet (gargle_options), 13
GceToken, 14

GceToken(), 6

httr, 19, 20
httr::config(), 4, 5, 17, 22
httr::init_oauth2.0(), 13
httr::oauth_app(), 8, 11, 12, 15
httr::Token, 23
httr::Token2.0, 2, 4
httr::TokenServiceAccount, 2, 4, 7
httr::user_agent(), 19

init_AuthState, 14
init_AuthState(), 3

jsonlite::fromJson(), 7, 16

oauth_app_from_json, 15

request_build (request_develop), 16
request_build(), 19, 20
request_develop, 16, 19, 21
request_develop(), 19
request_make, 18, 19, 21
request_make(), 20
response_as_json (response_process), 20
response_process, 18, 19, 20
response_process(), 19, 20

service_account token, 7

token-info, 22
Token2.0, 8, 5, 22
token_email (token-info), 22
token_fetch, 4–8, 23
token_fetch(), 2, 9, 10, 15
token_tokeninfo (token-info), 22
token_userinfo (token-info), 22