Package ‘gmailr’

June 30, 2023

Title Access the 'Gmail' 'RESTful' API
Version 2.0.0
Description An interface to the 'Gmail' 'RESTful' API. Allows access to your 'Gmail' messages, threads, drafts and labels.
License MIT + file LICENSE
BugReports https://github.com/r-lib/gmailr/issues
Depends R (>= 3.6)
Imports base64enc, cli, crayon, gargle (>= 1.5.1), glue, httr, jsonlite, lifecycle, mime, rappdirs, rematch2, rlang (>= 1.1.0)
Suggests curl, knitr, methods, rmarkdown, testthat (>= 3.1.8), withr
VignetteBuilder knitr
Config/Needs/website tidyverse/tidytemplate
Encoding UTF-8
RoxygenNote 7.2.3
Config/testthat/edition 3
NeedsCompilation no
Author Jim Hester [aut], Jennifer Bryan [aut, cre], Posit Software, PBC [cph, fnd]
Maintainer Jennifer Bryan <jenny@posit.co>
Repository CRAN
Date/Publication 2023-06-30 05:40:02 UTC

R topics documented:

as.character.mime .......................................................... 3
gmailr-configuration ..................................................... 3
gm_attachment ............................................................. 4
gm_attachments ......................................................... 5
gm_auth ................................................................. 6
gm_auth_configure ...................................................... 9
gm_body ................................................................. 10
gm_create_draft .......................................................... 11
gm_create_label .......................................................... 11
gm_deauth ............................................................... 12
gm_delete_draft .......................................................... 13
gm_delete_label .......................................................... 13
gm_delete_message ......................................................... 14
gm_delete_thread .......................................................... 15
gm_draft ................................................................. 15
gm_drafts ................................................................. 16
gm_has_token ............................................................. 17
gm_history .............................................................. 17
gm_id ................................................................. 18
gm_import_message ....................................................... 19
gm_insert_message ......................................................... 20
gm_label ................................................................. 21
gm_labels ................................................................. 21
gm_last_response ........................................................ 22
gm_message .............................................................. 22
gm_messages .............................................................. 23
gm_mime ................................................................. 24
gm_modify_message .......................................................... 26
gm_modify_thread .......................................................... 27
gm_profile .............................................................. 28
gm_save_attachment ......................................................... 28
gm_save_attachments ....................................................... 29
gm_scopes ............................................................... 30
gm_send_draft ............................................................. 31
gm_send_message .......................................................... 32
gm_thread ............................................................... 33
gm_threads ............................................................... 33
gm_to ................................................................. 34
gm_token ............................................................... 35
gm_token_write ........................................................... 36
gm_trash_message .......................................................... 37
gm_trash_thread .......................................................... 38
gm_untrash_message ......................................................... 39
gm_untrash_thread .......................................................... 39
gm_update_label ............................................................ 40
quoted_printable_encode .................................................. 41

Index 42
as.character.mime  

Convert a mime object to character representation

Description

This function converts a mime object into a character vector.

Usage

## S3 method for class 'mime'
as.character(x, newline = "\r\n", ...)

Arguments

x
  object to convert
newline
  value to use as newline character
...
  further arguments ignored

gmailr-configuration  

Configuring gmailr

Description

gmailr can be configured with various environment variables, which are accessed through wrapper functions that provide some additional smarts.

Usage

gm_default_email()
gm_default_oauth_client()

gm_default_email()

gm_default_email() returns the environment variable GMAILR_EMAIL, if it exists, and gargle::gargle_oauth_email(), otherwise.

gm_default_oauth_client()

gm_default_oauth_client() consults a specific set of locations, looking for the filepath for the JSON file that represents an OAuth client. This file can be downloaded from the APIs & Services section of the Google Cloud console https://console.cloud.google.com). The search unfolds like so:

- GMAILR_OAUTH_CLIENT environment variable: If defined, it is assumed to be the path to the target JSON file.
A `.json` file found in the directory returned by `rappdirs::user_data_dir("gmailr")`, whose filename uniquely matches the regular expression "client_secret.[.]json$".

- GMAILR_APP environment variable: This is supported for backwards compatibility, but it is preferable to store the JSON below `rappdirs::user_data_dir("gmailr")` or to store the path in the GMAILR_OAUTH_CLIENT environment variable.

Here's an inspirational snippet to move the JSON file you downloaded into the right place for auto-discovery by `gm_auth_configure()`:

```r
path_old <- "~/Downloads/client_secret_123-abc.apps.googleusercontent.com.json"
d <- fs::dir_create(rappdirs::user_data_dir("gmailr"), recurse = TRUE)
fs::file_move(path_old, d)
```

**See Also**

Since `gmailr` uses the gargle package to handle auth, gargle’s configuration is also relevant, which is mostly accomplished through options and associated accessor functions.

Other auth functions: `gm_auth_configure()`, `gm_auth()`, `gm_deauth()`, `gm_scopes()`

**Examples**

```r
gm_default_email()
withr::with_envvar(
  c(GMAILR_EMAIL = "jenny@example.com"),
  gm_default_email()
)

gm_default_oauth_client()
withr::with_envvar(
  c(GMAILR_OAUTH_CLIENT = "path/to/my-client.json"),
  gm_default_oauth_client()
)
```

---

**gm_attachment**

*Retrieve an attachment to a message*

**Description**

This is a low level function to retrieve an attachment to a message by id of the attachment and message. Most users are better off using `gm_save_attachments()` to automatically save all the attachments in a given message.

**Usage**

```r
gm_attachment(id, message_id, user_id = "me")
```
Arguments

<table>
<thead>
<tr>
<th>id</th>
<th>id of the attachment</th>
</tr>
</thead>
<tbody>
<tr>
<td>message_id</td>
<td>id of the parent message</td>
</tr>
<tr>
<td>user_id</td>
<td>gmail user_id to access, special value of 'me' indicates the authenticated user.</td>
</tr>
</tbody>
</table>

References

https://developers.google.com/gmail/api/reference/rest/v1/users.messages.attachments/get

See Also

Other message: gm_delete_message(), gm_import_message(), gm_insert_message(), gm_messages(), gm_message(), gm_modify_message(), gm_save_attachments(), gm_save_attachment(), gm_send_message(), gm_trash_message(), gm_untrash_message()

Examples

```r
## Not run:
my_attachment <- gm_attachment("a32e324b", "12345")
# save attachment to a file
gm_save_attachment(my_attachment, "photo.jpg")
## End(Not run)
```

---

**gm_attachments**

Retrieves information about attachments.

**Description**

Retrieve information about attachments.

**Usage**

```r
gm_attachments(x, ...)
```

**Arguments**

- x
  - An object from which to retrieve the attachment information.
- ...
  - other parameters passed to methods

**Value**

A data.frame with the filename, type, size and id of each attachment in the message.
gm_auth

Authorize gmailr

Description

Authorize gmailr to view and manage your Gmail projects. This function is a wrapper around `gargle::token_fetch()`.

By default, you are directed to a web browser, asked to sign in to your Google account, and to grant gmailr permission to operate on your behalf with Google Gmail. By default, with your permission, these user credentials are cached in a folder below your home directory, from where they can be automatically refreshed, as necessary. Storage at the user level means the same token can be used across multiple projects and tokens are less likely to be synced to the cloud by accident.

Usage

```r
gm_auth(
  email = gm_default_email(),
  path = NULL,
  subject = NULL,
  scopes = "full",
  cache = gargle::gargle_oauth_cache(),
  use_oob = gargle::gargle_oob_default(),
  token = NULL
)
```

Arguments

- `email` Optional. If specified, `email` can take several different forms:
  - "jane@gmail.com", i.e. an actual email address. This allows the 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).
  - "+@example.com", i.e. a domain-only glob pattern. This can be helpful if you need code that "just works" for both alice@example.com and bob@example.com.
  - TRUE means that you are approving email auto-discovery. If exactly one matching token is found in the cache, it will be used.
  - FALSE or NA mean that you want to ignore the token cache and force a new OAuth dance in the browser.

Defaults to the option named "gargle_oauth_email", retrieved by `gargle_oauth_email()` (unless a wrapper package implements different default behavior).

- `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).
subject
An optional subject claim. Specify this if you wish to use the service account represented by path to impersonate the subject, who is a normal user. Before this can work, an administrator must grant the service account domain-wide authority. Identify the user to impersonate via their email, e.g. subject = "user@example.com". Note that gargle automatically adds the non-sensitive "https://www.googleapis.com/auth/userinfo.email" scope, so this scope must be enabled for the service account, along with any other scopes being requested.

scopes
One or more API scopes. Each scope can be specified in full or, for Gmail API-specific scopes, in an abbreviated form that is recognized by gm_scopes():

- "full" = "https://mail.google.com/" (the default)
- "gmail.compose" = "https://www.googleapis.com/auth/gmail.compose"
- "gmail.readonly" = "https://www.googleapis.com/auth/gmail.readonly"
- "gmail.labels" = "https://www.googleapis.com/auth/gmail.labels"
- "gmail.send" = "https://www.googleapis.com/auth/gmail.send"
- "gmail.insert" = "https://www.googleapis.com/auth/gmail.insert"
- "gmail.modify" = "https://www.googleapis.com/auth/gmail.modify"
- "gmail.metadata" = "https://www.googleapis.com/auth/gmail.metadata"
- "gmail.labels" = "https://www.googleapis.com/auth/gmail.labels"
- "gmail.settings_basic" = "https://www.googleapis.com/auth/gmail.settings.basic"
- "gmail.settings_sharing" = "https://www.googleapis.com/auth/gmail.settings.sharing"

See [https://developers.google.com/gmail/api/auth/scopes](https://developers.google.com/gmail/api/auth/scopes) for details on the permissions for each scope.

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

use_oob
Whether to use out-of-band authentication (or, perhaps, a variant implemented by gargle and known as "pseudo-OOB") when first acquiring the token. Defaults to the value returned by gargle_oob_default(). Note that (pseudo-)OOB auth only affects the initial OAuth dance. If we retrieve (and possibly refresh) a cached token, use_oob has no effect.

If the OAuth client is provided implicitly by a wrapper package, its type probably defaults to the value returned by gargle_oauth_client_type(). You can take control of the client type by setting options(gargle_oauth_client_type = "web") or options(gargle_oauth_client_type = "installed").

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.

Details
Most users, most of the time, do not need to call gm_auth() explicitly – it is triggered by the first action that requires authorization. Even when called, the default arguments often suffice.

However, when necessary, gm_auth() allows the user to explicitly:

- Declare which Google identity to use, via an email specification.
- Use a service account token or workload identity federation via path.
gm_auth

- Bring your own token.
- Customize scopes.
- Use a non-default cache folder or turn caching off.
- Explicitly request out-of-bound (OOB) auth via use_oob.

If you are interacting with R within a browser (applies to RStudio Server, Posit Workbench, Posit Cloud, and Google Colaboratory), you need OOB auth or the pseudo-OOB variant. If this does not happen automatically, you can request it explicitly with use_oob = TRUE or, more persistently, by setting an option via options(gargle_oob_default = TRUE).

The choice between conventional OOB or pseudo-OOB auth is determined by the type of OAuth client. If the client is of the "installed" type, use_oob = TRUE results in conventional OOB auth. If the client is of the "web" type, use_oob = TRUE results in pseudo-OOB auth. Packages that provide a built-in OAuth client can usually detect which type of client to use. But if you need to set this explicitly, use the "gargle_oauth_client_type" option:

options(gargle_oauth_client_type = "web")  # pseudo-OOB
# or, alternatively
options(gargle_oauth_client_type = "installed")  # conventional OOB

For details on the many ways to find a token, see gargle::token_fetch(). For deeper control over auth, use gm_auth_configure() to bring your own OAuth client or API key. To learn more about gargle options, see gargle::gargle_options.

See Also

Other auth functions: gm_auth_configure(), gm_deauth(), gm_scopes(), gmailr-configuration

Examples

# load/refresh existing credentials, if available
# otherwise, go to browser for authentication and authorization
gm_auth()

# indicate the specific identity you want to auth as
gm_auth(email = "jenny@example.com")

# force a new browser dance, i.e. don't even try to use existing user
# credentials
gm_auth(email = NA)

# specify the identity, use a 'read only' scope, so it's impossible to
# send or delete email, and specify a cache folder
gm_auth(
  "target.user@example.com",
  scopes = "gmail.readonly",
  cache = "some/nice/directory/"
)


**gm_auth_configure**

*Edit auth configuration*

---

**Description**

See the article [Set up an OAuth client](#) for instructions on how to get an OAuth client. Then you can use `gm_auth_configure()` to register your client for use with gmailr. `gm_oauth_client()` retrieves the currently configured OAuth client.

**Usage**

```r
gm_auth_configure(
  client = NULL,
  path = gm_default_oauth_client(),
  key = deprecated(),
  secret = deprecated(),
  appname = deprecated(),
  app = deprecated()
)
```

**Arguments**

- **client**
  A Google OAuth client, presumably constructed via `gargle::gargle_oauth_client_from_json()`. Note, however, that it is preferred to specify the client with JSON, using the `path` argument.

- **path**
  JSON downloaded from Google Cloud Console, containing a client id and secret, in one of the forms supported for the `txt` argument of `jsonlite::fromJSON()` (typically, a file path or JSON string).

- **key, secret, appname, app**
  [Deprecated] Use the `path` (strongly recommended) or `client` argument instead.

**Value**

- `gm_auth_configure()`: An object of R6 class `gargle::AuthState`, invisibly.
- `gm_oauth_client()`: the current user-configured OAuth client.

**See Also**

- `gm_default_oauth_client()` to learn how you can make your OAuth client easy for gmailr to discover.
- Other auth functions: `gm_auth()`, `gm_deauth()`, `gm_scopes()`, `gmailr-configuration`
Examples

# if your OAuth client can be auto-discovered (see ?gm_default_oauth_client),
# you don't need to provide anything!
gm_auth_configure()

# see and store the current user-configured OAuth client
(original_client <- gm_oauth_client())

# the preferred way to configure your own client is via a JSON file
# downloaded from Google Developers Console
# this example JSON is indicative, but fake
path_to_json <- system.file(
  "extdata", "client_secret_installed.googleusercontent.com.json",
  package = "gargle"
)
gm_auth_configure(path = path_to_json)

# confirm that a (fake) OAuth client is now configured
gm_oauth_client()

# restore original auth config
gm_auth_configure(client = original_client)

---

**gm_body**

*Get the body text of a message or draft*

Description

Get the body text of a message or draft

Usage

gm_body(x, ...)

Arguments

x the object from which to retrieve the body

... other parameters passed to methods

Examples

```r
## Not run:
gm_body(my_message)
```

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

Create a draft from a mime message

**Description**

Create a draft from a mime message

**Usage**

```r
gm_create_draft(mail, user_id = "me")
```

**Arguments**

- `mail`: mime mail message created by mime
- `user_id`: gmail user_id to access, special value of 'me' indicates the authenticated user.

**References**

https://developers.google.com/gmail/api/reference/rest/v1/users.drafts/create

**Examples**

```r
## Not run:
gm_create_draft(gm_mime(
  From = "you@me.com", To = "any@one.com",
  Subject = "hello", "how are you doing?"
))
## End(Not run)
```

---

**gm_create_label**

Create a new label

**Description**

Function to create a label.

**Usage**

```r
gm_create_label(
  name,
  label_list_visibility = c("show", "hide", "show_unread"),
  message_list_visibility = c("show", "hide"),
  user_id = "me"
)
```
**Arguments**

<table>
<thead>
<tr>
<th>name</th>
<th>Argument Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name to give to the new label</td>
</tr>
<tr>
<td>label_list_visibility</td>
<td>The visibility of the label in the label list in the Gmail web interface.</td>
</tr>
<tr>
<td>message_list_visibility</td>
<td>The visibility of messages with this label in the message list in the Gmail web interface.</td>
</tr>
<tr>
<td>user_id</td>
<td>gmail user_id to access, special value of 'me' indicates the authenticated user.</td>
</tr>
</tbody>
</table>

**References**

https://developers.google.com/gmail/api/reference/rest/v1/users.labels/create

**See Also**

Other label: gm_delete_label(), gm_labels(), gm_label(), gm_update_label()

---

**gm_deauth**

*Clear current token*

**Description**

Clears any currently stored token. The next time gmailr needs a token, the token acquisition process starts over, with a fresh call to gm_auth() and, therefore, internally, a call to gargle::token_fetch(). Unlike some other packages that use gargle, gmailr is not usable in a de-authorized state. Therefore, calling gm_deauth() only clears the token, i.e. it does NOT imply that subsequent requests are made with an API key in lieu of a token.

**Usage**

gm_deauth()

**See Also**

Other auth functions: gm_auth_configure(), gm_auth(), gm_scopes(), gmailr-configuration

**Examples**

gm_deauth()
**gm_delete_draft**

Permanently delete a single draft

**Description**

Function to delete a given draft by id. This cannot be undone!

**Usage**

\[\text{gm_delete_draft}(\text{id}, \text{user_id} = "me")\]

**Arguments**

- **id**: message id to access
- **user_id**: gmail user_id to access, special value of ‘me’ indicates the authenticated user.

**References**

https://developers.google.com/gmail/api/reference/rest/v1/users.drafts/delete

**See Also**

Other draft: gm_drafts(), gm_draft(), gm_send_draft()

**Examples**

```r
## Not run:
date("12345")

## End(Not run)
```

---

**gm_delete_label**

Permanently delete a label

**Description**

Function to delete a label by id. This cannot be undone!

**Usage**

\[\text{gm_delete_label}(\text{id}, \text{user_id} = "me")\]

**Arguments**

- **id**: label id to retrieve
- **user_id**: gmail user_id to access, special value of ‘me’ indicates the authenticated user.
gm_delete_message

References

https://developers.google.com/gmail/api/reference/rest/v1/users.labels/delete

See Also

Other label: gm_create_label(), gm_labels(), gm_label(), gm_update_label()

---

**gm_delete_message**  
Permanently delete a single message

---

**Description**

Function to delete a given message by id. This cannot be undone!

**Usage**

```r
gm_delete_message(id, user_id = "me")
```

**Arguments**

- **id**  
message id to access

- **user_id**  
gmail user id to access, special value of 'me' indicates the authenticated user.

**References**

https://developers.google.com/gmail/api/reference/rest/v1/users.messages/delete

See Also

Other message: gm_attachment(), gm_import_message(), gm_insert_message(), gm_messages(), gm_message(), gm_modify_message(), gm_save_attachments(), gm_save_attachment(), gm_send_message(), gm_trash_message(), gm_untrash_message()

**Examples**

```r
## Not run:
gm_delete_message("12345")

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

*Permanently delete a single thread.*

**Description**
Function to delete a given thread by id. This cannot be undone!

**Usage**

```r
gm_delete_thread(id, user_id = "me")
```

**Arguments**

- `id` thread id to access
- `user_id` gmail user_id to access, special value of 'me' indicates the authenticated user.

**References**

https://developers.google.com/gmail/api/reference/rest/v1/users.threads/delete

**See Also**
Other thread: `gm_modify_thread()`, `gm_threads()`, `gm_thread()`, `gm_trash_thread()`, `gm_untrash_thread()`

**Examples**

```r
## Not run:
gm_delete_thread(12345)
## End(Not run)
```

---

**gm_draft**

*Get a single draft*

**Description**
Function to retrieve a given draft by <-

**Usage**

```r
gm_draft(id, user_id = "me", format = c("full", "minimal", "raw"))
```

**Arguments**

- `id` draft id to access
- `user_id` gmail user_id to access, special value of 'me' indicates the authenticated user.
- `format` format of the draft returned
gm_drafts

Get a list of drafts

Description

Get a list of drafts possibly matching a given query string.

Usage

gm_drafts(num_results = NULL, page_token = NULL, user_id = "me")

Arguments

num_results the number of results to return.
page_token retrieve a specific page of results
user_id gmail user_id to access, special value of 'me' indicates the authenticated user.

References

https://developers.google.com/gmail/api/reference/rest/v1/users.drafts/list

See Also

Other draft: gm_delete_draft(), gm_drafts(), gm_send_draft()

Examples

## Not run:
my_draft <- gm_draft("12345")

## End(Not run)

my_drafts <- gm_drafts()
first_10_drafts <- gm_drafts(10)

## End(Not run)
**gm_has_token**

Is there a token on hand?

**Description**

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

**Usage**

```r
gm_has_token()
```

**Value**

Logical.

**See Also**

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

**Examples**

```r
gm_has_token()
```

---

**gm_history**

Retrieve change history for the inbox

**Description**

Retrieves the history results in chronological order

**Usage**

```r
gm_history(
  start_history_id = NULL,
  num_results = NULL,
  label_id = NULL,
  page_token = NULL,
  user_id = "me"
)
```
Arguments

- **start_history_id**: the point to start the history. The historyId can be obtained from a message, thread or previous list response.
- **num_results**: the number of results to return, max per page is 100
- **label_id**: filter history only for this label
- **page_token**: retrieve a specific page of results
- **user_id**: gmail user_id to access, special value of 'me' indicates the authenticated user.

References

https://developers.google.com/gmail/api/reference/rest/v1/users.history/list

Examples

```r
## Not run:
my_history <- gm_history("10")
## End(Not run)
```

---

**gm_id**

*Get the id of a gmailr object*

Description

Get the id of a gmailr object

Usage

```r
gm_id(x, ...)
```

## S3 method for class 'gmail_messages'
```r
gm_id(x, what = c("message_id", "thread_id"), ...)
```

Arguments

- **x**: the object from which to retrieve the id
- **...**: other parameters passed to methods
- **what**: the type of id to return

Examples

```r
## Not run:
gm_id(my_message)
gm_id(my_draft)
## End(Not run)
```
Import a message into the gmail mailbox from a mime message

Description

Import a message into the gmail mailbox from a mime message

Usage

\[
\text{gm\_import\_message}( \\
\quad \text{mail,} \\
\quad \text{label\_ids,} \\
\quad \text{type = c("multipart", "media", "resumable"),} \\
\quad \text{internal\_date\_source = c("dateHeader", "recievedTime"),} \\
\quad \text{user\_id = "me"} \\
\)
\]

Arguments

- **mail**: mime mail message created by mime
- **label_ids**: optional label ids to apply to the message
- **type**: the type of upload to perform
- **internal_date_source**: whether to date the object based on the date of the message or when it was received by gmail.
- **user_id**: gmail user_id to access, special value of 'me' indicates the authenticated user.

References

[https://developers.google.com/gmail/api/reference/rest/v1/users.messages/import](https://developers.google.com/gmail/api/reference/rest/v1/users.messages/import)

See Also

Other message: \text{gm\_attachment()}, \text{gm\_delete\_message()}, \text{gm\_insert\_message()}, \text{gm\_messages()}, \text{gm\_message()}, \text{gm\_modify\_message()}, \text{gm\_save\_attachments()}, \text{gm\_save\_attachment()}, \text{gm\_send\_message()}, \text{gm\_trash\_message()}, \text{gm\_untrash\_message()}

Examples

## Not run:

\[
\text{gm\_import\_message(gm\_mime(} \\
\quad \text{From = "you@me.com", To = "any@one.com",} \\
\quad \text{Subject = "hello", "how are you doing?"}) \\
\]

## End(Not run)
gm_insert_message

Insert a message into the gmail mailbox from a mime message

Description

Insert a message into the gmail mailbox from a mime message

Usage

gm_insert_message(
  mail,
  label_ids,
  type = c("multipart", "media", "resumable"),
  internal_date_source = c("dateHeader", "recievedTime"),
  user_id = "me"
)

Arguments

mail         mime mail message created by mime
label_ids    optional label ids to apply to the message
type         the type of upload to perform
internal_date_source
              whether to date the object based on the date of the message or when it was
              received by gmail.
user_id      gmail user_id to access, special value of 'me' indicates the authenticated user.

References

https://developers.google.com/gmail/api/reference/rest/v1/users.messages/insert

See Also

Other message: gm_attachment(), gm_delete_message(), gm_import_message(), gm_messages(),
gm_message(), gm_modify_message(), gm_save_attachments(), gm_save_attachment(), gm_send_message(),
gm_trash_message(), gm_untrash_message()

Examples

## Not run:
gm_insert_message(gm_mime(
  From = "you@me.com", To = "any@one.com",
  Subject = "hello", "how are you doing?"
))

## End(Not run)
**gm_label**  
*Get a specific label*

**Description**
Get a specific label by id and user_id.

**Usage**
```
gm_label(id, user_id = "me")
```

**Arguments**
- `id`: label id to retrieve
- `user_id`: gmail user_id to access, special value of 'me' indicates the authenticated user.

**References**
[https://developers.google.com/gmail/api/reference/rest/v1/users.labels/get](https://developers.google.com/gmail/api/reference/rest/v1/users.labels/get)

**See Also**
Other label: `gm_create_label()`, `gm_delete_label()`, `gm_labels()`, `gm_update_label()`

---

**gm_labels**  
*Get a list of all labels*

**Description**
Get a list of all labels for a user.

**Usage**
```
gm_labels(user_id = "me")
```

**Arguments**
- `user_id`: gmail user_id to access, special value of 'me' indicates the authenticated user.

**References**
[https://developers.google.com/gmail/api/reference/rest/v1/users.labels/list](https://developers.google.com/gmail/api/reference/rest/v1/users.labels/list)

**See Also**
Other label: `gm_create_label()`, `gm_delete_label()`, `gm_label()`, `gm_update_label()`
Examples

```r
## Not run:
my_labels <- gm_labels()

## End(Not run)
```

---

**gm_last_response**  
Response from the last query

**Usage**

```r
gm_last_response()
```

---

**gm_message**  
Get a single message

**Description**

Function to retrieve a given message by id

**Usage**

```r
gm_message(
  id,
  user_id = "me",
  format = c("full", "metadata", "minimal", "raw")
)
```

**Arguments**

- `id`: message id to access
- `user_id`: gmail user_id to access, special value of 'me' indicates the authenticated user.
- `format`: format of the message returned

**References**

[https://developers.google.com/gmail/api/reference/rest/v1/users.messages](https://developers.google.com/gmail/api/reference/rest/v1/users.messages)

**See Also**

Other message: `gm_attachment()`, `gm_delete_message()`, `gm_import_message()`, `gm_insert_message()`, `gm_messages()`, `gm_modify_message()`, `gm_save_attachments()`, `gm_save_attachment()`, `gm_send_message()`, `gm_trash_message()`, `gm_untrash_message()`
Examples

## Not run:
my_message <- gm_message(12345)

## End(Not run)

---

**gm_messages**

*Get a list of messages*

**Description**

Get a list of messages possibly matching a given query string.

**Usage**

```r
gm_messages(
  search = NULL,
  num_results = NULL,
  label_ids = NULL,
  include_spam_trash = NULL,
  page_token = NULL,
  user_id = "me"
)
```

**Arguments**

- **search**: query to use, same format as gmail search box.
- **num_results**: the number of results to return.
- **label_ids**: restrict search to given labels
- **include_spam_trash**: boolean whether to include the spam and trash folders in the search
- **page_token**: retrieve a specific page of results
- **user_id**: gmail user_id to access, special value of 'me' indicates the authenticated user.

**References**

https://developers.google.com/gmail/api/reference/rest/v1/users.messages/list

**See Also**

Other message: `gm_attachment()`, `gm_delete_message()`, `gm_import_message()`, `gm_insert_message()`, `gm_message()`, `gm_modify_message()`, `gm_save_attachments()`, `gm_save_attachment()`, `gm_send_message()`, `gm_trash_message()`, `gm_untrash_message()`
Examples

```r
## Not run:
# Search for R, return 10 results using label 1 including spam and trash folders
my_messages <- gm_messages("R", 10, "label_1", TRUE)
## End(Not run)
```

---

**gm_mime**  
Create a mime formatted message object

### Description
These functions create a MIME message. They can be created atomically using `gm_mime()` or iteratively using the various accessors.

### Usage

```r
gm_mime(..., attr = NULL, body = NULL, parts = list())
```

#### S3 method for class 'mime'

```r
gm_to(x, val, ...)
```

#### S3 method for class 'mime'

```r
gm_from(x, val, ...)
```

#### S3 method for class 'mime'

```r
gm_cc(x, val, ...)
```

#### S3 method for class 'mime'

```r
gm_bcc(x, val, ...)
```

#### S3 method for class 'mime'

```r
gm_subject(x, val, ...)
```

```r
gm_text_body(
  mime,
  body,
  content_type = "text/plain",
  charset = "utf-8",
  encoding = "quoted-printable",
  format = "flowed",
  ...
)
```

```r
gm_html_body(
  mime,
  body,
  ...)
gm_mime

content_type = "text/html",
charset = "utf-8",
encoding = "base64",
...
)

gm_attach_part(mime, part, id = NULL, ...)

gm_attach_file(mime, filename, type = NULL, id = NULL, ...)

Arguments

... additional parameters to put in the attr field
attr attributes to pass to the message
body Message body.
parts mime parts to pass to the message
x the object whose fields you are setting
val the value to set, can be a vector, in which case the values will be joined by ",",.
mime message.
content_type The content type to use for the body.
charset The character set to use for the body.
encoding The transfer encoding to use for the body.
format The mime format to use for the body.
part Message part to attach
id The content ID of the attachment
filename name of file to attach
type mime type of the attached file

Examples

# using the field functions
msg <- gm_mime() |>
  gm_from("james.f.hestergmail.com") |>
  gm_to("asdfasdf.com") |>
  gm_text_body("Test Message")

# alternatively you can set the fields using gm_mime(), however you have
# to use properly formatted MIME names
msg <- gm_mime(
  From = "james.f.hestergmail.com",
  To = "asdfasdf.com",
) |>
  gm_html_body("<b>Test</b> Message")
gm_modify_message

Modify the labels on a message

Description

Function to modify the labels on a given message by id. Note you need to use the label ID as arguments to this function, not the label name.

Usage

gm_modify_message(id, add_labels = NULL, remove_labels = NULL, user_id = "me")

Arguments

id          message id to access
add_labels  label IDs to add to the specified message
remove_labels label IDs to remove from the specified message
user_id     gmail user_id to access, special value of 'me' indicates the authenticated user.

References

https://developers.google.com/gmail/api/reference/rest/v1/users.messages/modify

See Also

Other message: gm_attachment(), gm_delete_message(), gm_import_message(), gm_insert_message(), gm_messages(), gm_message(), gm_save_attachments(), gm_save_attachment(), gm_send_message(), gm_trash_message(), gm_untrash_message()

Examples

## Not run:
gm_modify_message(12345, add_labels = "label_1")
gm_modify_message(12345, remove_labels = "label_1")
# add and remove at the same time
gm_modify_message(12345, add_labels = "label_2", remove_labels = "label_1")

## End(Not run)
**gm_modify_thread**  
Modify the labels on a thread

**Description**
Function to modify the labels on a given thread by id.

**Usage**

```r
gm_modify_thread(  
  id,  
  add_labels = character(0),  
  remove_labels = character(0),  
  user_id = "me"  
)
```

**Arguments**

- **id**: thread id to access
- **add_labels**: labels to add to the specified thread
- **remove_labels**: labels to remove from the specified thread
- **user_id**: gmail user_id to access, special value of 'me' indicates the authenticated user.

**References**

[https://developers.google.com/gmail/api/reference/rest/v1/users.threads/modify](https://developers.google.com/gmail/api/reference/rest/v1/users.threads/modify)

**See Also**

Other thread: `gm_delete_thread()`, `gm_threads()`, `gm_thread()`, `gm_trash_thread()`, `gm_untrash_thread()`

**Examples**

```r
## Not run:
gm_modify_thread(12345, add_labels = "label_1")
gm_modify_thread(12345, remove_labels = "label_1")
# add and remove at the same time
gm_modify_thread(12345, add_labels = "label_2", remove_labels = "label_1")
## End(Not run)
```
**gm_profile**  
*Get info on current gmail profile*

**Description**
Reveals information about the profile associated with the current token.

**Usage**
```
gm_profile(user_id = "me", verbose = TRUE)
```

**Arguments**
- **user_id**  
gmail user_id to access, special value of 'me' indicates the authenticated user.
- **verbose**  
Logical, indicating whether to print informative messages (default TRUE).

**Value**
A list of class `gmail_profile`.

**See Also**
Wraps the `getProfile` endpoint:
- [https://developers.google.com/gmail/api/reference/rest/v1/users/getProfile](https://developers.google.com/gmail/api/reference/rest/v1/users/getProfile)

**Examples**
```r
## Not run:
gm_profile()
## more info is returned than is printed
prof <- gm_profile()
prof["historyId"]

## End(Not run)
```

**gm_save_attachment**  
*Save the attachment to a file*

**Description**
This is a low level function that only works on attachments retrieved with `gm_attachment()`. To save an attachment directly from a message see `gm_save_attachments()`, which is a higher level interface more suitable for most uses.
Usage

```r
gm_save_attachment(x, filename)
```

Arguments

- `x` : attachment to save
- `filename` : location to save to

See Also

Other message: `gm_attachment()`, `gm_delete_message()`, `gm_import_message()`, `gm_insert_message()`, `gm_messages()`, `gm_message()`, `gm_modify_message()`, `gm_save_attachments()`, `gm_send_message()`, `gm_trash_message()`, `gm_untrash_message()`

Examples

```r
## Not run:
my_attachment <- gm_attachment("a32e324b", "12345")
# save attachment to a file
gm_save_attachment(my_attachment, "photo.jpg")
## End(Not run)
```

---

**gm_save_attachments**  
Save attachments to a message

Description

Function to retrieve and save all of the attachments to a message by id of the message.

Usage

```r
gm_save_attachments(x, attachment_id = NULL, path = ".", user_id = "me")
```

Arguments

- `x` : message with attachment
- `attachment_id` : id of the attachment to save, if none specified saves all attachments
- `path` : where to save the attachments
- `user_id` : gmail user_id to access, special value of 'me' indicates the authenticated user.

References

https://developers.google.com/gmail/api/reference/rest/v1/users.messages.attachments/get
See Also

Other message: `gm_attachment()`, `gm_delete_message()`, `gm_import_message()`, `gm_insert_message()`, `gm_messages()`, `gm_message()`, `gm_modify_message()`, `gm_save_attachment()`, `gm_send_message()`, `gm_trash_message()`, `gm_untrash_message()`

Examples

```r
## Not run:
# save all attachments
gm_save_attachments(my_message)
# save a specific attachment
gm_save_attachments(my_message, "a32e324b")

## End(Not run)
```

---

**gm_scopes**

Produce scopes specific to the Gmail API

Description

When called with no arguments, `gm_scopes()` returns a named character vector of scopes associated with the Gmail API. If `gm_scopes(scopes =)` is given, an abbreviated entry such as "gmail.readonly" is expanded to a full scope ("https://www.googleapis.com/auth/gmail.readonly" in this case). Unrecognized scopes are passed through unchanged.

Usage

`gm_scopes(scopes = NULL)`

Arguments

- `scopes` One or more API scopes. Each scope can be specified in full or, for Gmail API-specific scopes, in an abbreviated form that is recognized by `gm_scopes()`:
  - "full" = "https://mail.google.com/" (the default)
  - "gmail.compose" = "https://www.googleapis.com/auth/gmail.compose"
  - "gmail.readonly" = "https://www.googleapis.com/auth/gmail.readonly"
  - "gmail.labels" = "https://www.googleapis.com/auth/gmail.labels"
  - "gmail.send" = "https://www.googleapis.com/auth/gmail.send"
  - "gmail.insert" = "https://www.googleapis.com/auth/gmail.insert"
  - "gmail.modify" = "https://www.googleapis.com/auth/gmail.modify"
  - "gmail.metadata" = "https://www.googleapis.com/auth/gmail.metadata"
  - "gmail.settings_basic" = "https://www.googleapis.com/auth/gmail.settings.basic"
  - "gmail.settings_sharing" = "https://www.googleapis.com/auth/gmail.settings.sharing"

See [https://developers.google.com/gmail/api/auth/scopes](https://developers.google.com/gmail/api/auth/scopes) for details on the permissions for each scope.
gm_send_draft

Value

A character vector of scopes.

See Also

https://developers.google.com/gmail/api/auth/scopes for details on the permissions for each scope.

Other auth functions: gm_auth_configure(), gm_auth(), gm_deauth(), gmailr-configuration

Examples

gm_scopes("full")
gm_scopes("gmail.readonly")
gm_scopes()

Description

Send a draft to the recipients in the To, CC, and Bcc headers.

Usage

gm_send_draft(draft, user_id = "me")

Arguments

draft the draft to send
user_id gmail user_id to access, special value of 'me' indicates the authenticated user.

References

https://developers.google.com/gmail/api/reference/rest/v1/users.drafts/send

See Also

Other draft: gm_delete_draft(), gm_drafts(), gm_draft()

Examples

## Not run:
draft <- gm_create_draft(gm_mime(
  From = "you@me.com", To = "any@one.com",
  Subject = "hello", "how are you doing?"
))
gm_send_draft(draft)

## End(Not run)
gm_send_message

Send a message from a mime message

Description

Send a message from a mime message

Usage

```r
gm_send_message(
    mail,  
    type = c("multipart", "media", "resumable"),
    thread_id = NULL,
    user_id = "me"
)
```

Arguments

- `mail`: mime mail message created by `mime`
- `type`: the type of upload to perform
- `thread_id`: the id of the thread to send from.
- `user_id`: gmail user_id to access, special value of 'me' indicates the authenticated user.

References

https://developers.google.com/gmail/api/reference/rest/v1/users.messages/send

See Also

Other message: `gm_attachment()`, `gm_delete_message()`, `gm_import_message()`, `gm_insert_message()`, `gm_messages()`, `gm_message()`, `gm_modify_message()`, `gm_save_attachments()`, `gm_save_attachment()`, `gm_trash_message()`, `gm_untrash_message()`

Examples

```r
## Not run:
gm_send_message(gm_mime(
    from = "you@me.com", to = "anyone.com",
    subject = "hello", "how are you doing?"
))

## End(Not run)
```
gm_thread

Get a single thread

Description

Function to retrieve a given thread by id

Usage

    gm_thread(id, user_id = "me")

Arguments

    id       thread id to access
    user_id  gmail user_id to access, special value of 'me' indicates the authenticated user.

References

    https://developers.google.com/gmail/api/reference/rest/v1/users.threads

See Also

Other thread: gm_delete_thread(), gm_modify_thread(), gm_threads(), gm_trash_thread(), gm_untrash_thread()

Examples

    ## Not run:
    my_thread <- gm_thread(12345)

    ## End(Not run)

gm_threads

Get a list of threads

Description

Get a list of threads possibly matching a given query string.
Usage

```r
gm_threads(
  search = NULL,
  num_results = NULL,
  page_token = NULL,
  label_ids = NULL,
  include_spam_trash = NULL,
  user_id = "me"
)
```

Arguments

- **search**: query to use, same format as gmail search box.
- **num_results**: the number of results to return.
- **page_token**: retrieve a specific page of results
- **label_ids**: restrict search to given labels
- **include_spam_trash**: boolean whether to include the spam and trash folders in the search
- **user_id**: gmail user_id to access, special value of 'me' indicates the authenticated user.

References

https://developers.google.com/gmail/api/reference/rest/v1/users.threads/list

See Also

Other thread: `gm_delete_thread()`, `gm_modify_thread()`, `gm_thread()`, `gm_trash_thread()`, `gm_untrash_thread()`

Examples

```r
## Not run:
my_threads <- gm_threads()
first_10_threads <- gm_threads(10)
## End(Not run)
```

Description

Methods to get values from message or drafts
Usage

\[ \text{gm\_to(x, ...)} \]
\[ \text{gm\_from(x, ...)} \]
\[ \text{gm\_cc(x, ...)} \]
\[ \text{gm\_bcc(x, ...)} \]
\[ \text{gm\_date(x, ...)} \]
\[ \text{gm\_subject(x, ...)} \]

Arguments

\[ x \quad \text{the object from which to get or set the field} \]
\[ ... \quad \text{other parameters passed to methods} \]

---

**gm\_token**

*Produce configured token*

Description

For internal use or for those programming around the Gmail 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, `gm\_auth()` is what they need. If there is no current token, `gm\_auth()` is called to either load from cache or initiate OAuth2.0 flow. If auth has been deactivated via `gm\_deauth()`, `gm\_token()` returns NULL.

Usage

`gm\_token()`

Value

A request object (an S3 class provided by `httr`).

See Also

Other low-level API functions: `gm\_has\_token()`

Examples

`gm\_token()`
**gm_token_write**

Write/read a gsamlr user token

**Description**

**[Experimental]**

This pair of functions writes an OAuth2 user token to file and reads it back in. This is rarely necessary when working in your primary, interactive computing environment. In that setting, it is recommended to lean into the automatic token caching built-in to gsamlr / gargle. However, when preparing a user token for use elsewhere, such as in CI or in a deployed data product, it can be useful to take the full control granted by `gm_token_write()` and `gm_token_read()`.

Below is an outline of the intended workflow, but you will need to fill in particulars, such as filepaths and environment variables:

1. Do auth in your primary, interactive environment as the target user, with the desired OAuth client and scopes.
   
   ```r
   gm_auth_configure()
   gm_auth("jane@example.com", cache = FALSE)
   ```

2. Confirm you are logged in as the intended user:
   
   ```r
   gm_profile()
   ```

3. Write the current token to file:
   
   ```r
   gm_token_write(
     path = "path/to/gsamlr-token.rds",
     key = "GMAILR_KEY"
   )
   ```

4. In the deployed, non-interactive setting, read the token from file and tell gsamlr to use it:
   
   ```r
   gm_auth(token = gm_token_read(
     path = "path/to/gsamlr-token.rds",
     key = "GMAILR_KEY"
   ))
   ```

**Usage**

```r
gm_token_write(token = gm_token(), path = "gsamlr-token.rds", key = NULL)

gm_token_read(path = "gsamlr-token.rds", key = NULL)
```
**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.
- **path**: The path to write to (`gm_token_write()`) or to read from (`gm_token_read()`).
- **key**: Encryption key, as implemented by `httr2`'s `secret functions`. If absent, a built-in key is used. If supplied, the key should usually be the name of an environment variable whose value was generated with `gargle::secret_make_key()` (which is a copy of `httr2::secret_make_key()`). The key argument of `gm_token_read()` must match the key used in `gm_token_write()`.

**Security**

`gm_token_write()` and `gm_token_read()` have a more security-oriented implementation than the default token caching strategy. OAuth2 user tokens are somewhat opaque by definition, because they aren’t written to file in a particularly transparent format. However, `gm_token_write()` always applies some additional obfuscation to make such credentials even more resilient against scraping by an automated tool. However, a knowledgeable R programmer could decode the credential with some effort. The default behaviour of `gm_token_write()` (called without key) is suitable for tokens stored in a relatively secure place, such as on Posit Connect within your organization.

To prepare a stored credential for exposure in a more public setting, such as on GitHub or CRAN, you must actually encrypt it, using a key known only to you. You must make the encryption key available via a secure environment variable in any setting where you wish to decrypt and use the token, such as on GitHub Actions.

---

**gm_trash_message**  
*Send a single message to the trash*

**Description**

Function to trash a given message by id. This can be undone by `gm_untrash_message()`.

**Usage**

```r
gm_trash_message(id, user_id = "me")
```

**Arguments**

- **id**: message id to access
- **user_id**: gmail user_id to access, special value of 'me' indicates the authenticated user.

**References**

[https://developers.google.com/gmail/api/reference/rest/v1/users.messages/trash](https://developers.google.com/gmail/api/reference/rest/v1/users.messages/trash)
See Also

Other message: `gm_attachment()`, `gm_delete_message()`, `gm_import_message()`, `gm_insert_message()`, `gm_messages()`, `gm_message()`, `gm_modify_message()`, `gm_save_attachments()`, `gm_save_attachment()`, `gm_send_message()`, `gm_untrash_message()`

Examples

```r
## Not run:
gm_trash_message("12345")

## End(Not run)
```

---

**gm_trash_thread**

Send a single thread to the trash

Description

Function to trash a given thread by id. This can be undone by `gm_untrash_thread()`.

Usage

```
gm_trash_thread(id, user_id = "me")
```

Arguments

- `id`: thread id to access
- `user_id`: gmail user_id to access, special value of `me` indicates the authenticated user.

References

https://developers.google.com/gmail/api/reference/rest/v1/users.threads/trash

See Also

Other thread: `gm_delete_thread()`, `gm_modify_thread()`, `gm_threads()`, `gm_thread()`, `gm_untrash_thread()`

Examples

```r
## Not run:
gm_trash_thread(12345)

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

Remove a single message from the trash.

**Description**

Function to trash a given message by id. This can be undone by `gm_untrash_message()`.

**Usage**

```r
gm_untrash_message(id, user_id = "me")
```

**Arguments**

- `id` message id to access
- `user_id` gmail user_id to access, special value of 'me' indicates the authenticated user.

**References**

[https://developers.google.com/gmail/api/reference/rest/v1/users.messages/trash](https://developers.google.com/gmail/api/reference/rest/v1/users.messages/trash)

**See Also**

Other message: `gm_attachment()`, `gm_delete_message()`, `gm_import_message()`, `gm_insert_message()`, `gm_messages()`, `gm_message()`, `gm_modify_message()`, `gm_save_attachments()`, `gm_save_attachment()`, `gm_send_message()`, `gm_trash_message()`

**Examples**

```r
## Not run:
gm_untrash_message("12345")

## End(Not run)
```

### gm_untrash_thread

Remove a single thread from the trash.

**Description**

Function to untrash a given thread by id. This can reverse the results of a previous `gm_trash_thread()`.

**Usage**

```r
gm_untrash_thread(id, user_id = "me")
```
**gm_update_label**

Update a existing label.

**Description**

Get a specific label by id and user_id. `gm_update_label_patch()` is identical to `gm_update_label()` but the latter uses HTTP PATCH to allow partial update.

**Usage**

```r
gm_update_label(id, label, user_id = "me")
gm_update_label_patch(id, label, user_id = "me")```

**Arguments**

- `id` : label id to update
- `label` : the label fields to update
- `user_id` : gmail user_id to access, special value of ’me’ indicates the authenticated user.

**References**

- https://developers.google.com/gmail/api/reference/rest/v1/users.labels/update
- https://developers.google.com/gmail/api/reference/rest/v1/users.labels/patch

**See Also**

Other label: `gm_create_label()`, `gm_delete_label()`, `gm_labels()`, `gm_label()`
quoted_printable_encode

Encode text using quoted printable

Description
Does no do any line wrapping of the output to 76 characters Implementation derived from the perl MIME::QuotedPrint

Usage
quoted_printable_encode(data)

Arguments
data data to encode

References
http://search.cpan.org/~gaas/MIME-Base64-3.14/QuotedPrint.pm
Index

* auth functions
  gm_auth, 6
  gm_auth_configure, 9
  gm_deauth, 12
  gm_scopes, 30
  gmailr-configuration, 3

* draft
  gm_delete_draft, 13
  gm_draft, 15
  gm_drafts, 16
  gm_send_draft, 31

* label
  gm_create_label, 11
  gm_delete_label, 13
  gm_label, 21
  gm_labels, 21
  gm_update_label, 40

* low-level API functions
  gm_has_token, 17
  gm_token, 35

* message
  gm_attachment, 4
  gm_delete_message, 14
  gm_import_message, 19
  gm_insert_message, 20
  gm_message, 22
  gm_messages, 23
  gm_modify_message, 26
  gm_save_attachment, 28
  gm_save_attachments, 29
  gm_send_message, 32
  gm_trash_message, 37
  gm_untrash_message, 39

* mime
  gm_mime, 24

* thread
  gm_delete_thread, 15
  gm_modify_thread, 27
  gm_thread, 33

  gm_threads, 33
  gm_trash_thread, 38
  gm_untrash_thread, 39

as.character.mime, 3

gargle::AuthState, 9

gargle::gargle_oauth_client_from_json(), 9

gargle::gargle_oauth_email(), 3

gargle::gargle_options, 8

gargle::token_fetch(), 6, 8, 12

gargle_oauth_cache(), 7

gargle_oauth_client_type(), 7

gargle_oauth_email(), 6

gargle_oob_default(), 7

  gm_attach_file(gm_mime), 24
  gm_attach_part(gm_mime), 24
  gm_attachment, 4, 14, 19, 20, 22, 23, 26, 29, 30, 32, 38, 39

  gm_attachment(), 28
  gm_attachments, 5
  gm_auth, 4, 6, 9, 12, 31
  gm_auth(), 12, 35
  gm_auth_configure, 4, 8, 9, 12, 31
  gm_authConfigure(), 8

  gm_bcc(gm_to), 34
  gm_bcc.mime(gm_mime), 24
  gm_body, 10
  gm_cc(gm_to), 34
  gm_cc.mime(gm_mime), 24
  gm_create_draft, 11
  gm_create_label, 11, 14, 21, 40
  gm_date(gm_to), 34
  gm_deauth, 4, 8, 9, 12, 31
  gm_deauth(), 35

  gm_default_email
    (gmailr-configuration), 3
  gm_default_oauth_client
    (gmailr-configuration), 3
INDEX

```
gm_default_oauth_client(), 9
gm_delete_draft, 13, 16, 31
gm_delete_label, 12, 13, 21, 40
gm_delete_message, 5, 14, 19, 20, 22, 23, 26, 29, 30, 32, 38, 39
gm_delete_thread, 15, 27, 33, 34, 38, 40
gm_draft, 13, 15, 16, 31
gm_drafts, 13, 16, 16, 31
gm_from (gm_to), 34
gm_from.mime (gm_mime), 24
gm_has_token, 17, 35
gm_history, 17
gm_html_body (gm_mime), 24
gm_id, 18
gm_import_message, 5, 14, 19, 20, 22, 23, 26, 29, 30, 32, 38, 39
gm_insert_message, 5, 14, 19, 20, 22, 23, 26, 29, 30, 32, 38, 39
gm_label, 12, 14, 21, 21, 40
gm_labels, 12, 14, 21, 21, 40
gm_last_response, 22
gm_message, 5, 14, 19, 20, 22, 23, 26, 29, 30, 32, 38, 39
gm_messages, 5, 14, 19, 20, 22, 23, 26, 29, 30, 32, 38, 39
gm_mime, 24
gm_modify_message, 5, 14, 19, 20, 22, 23, 26, 29, 30, 32, 38, 39
gm_modify_thread, 15, 27, 33, 34, 38, 40
gm_oauth_client (gm_auth_configure), 9
gm_profile, 28
gm_save_attachment, 5, 14, 19, 20, 22, 23, 26, 28, 30, 32, 38, 39
gm_save_attachments, 5, 14, 19, 20, 22, 23, 26, 29, 29, 32, 38, 39
gm_save_attachments(), 4, 28
gm_scopes, 4, 8, 9, 12, 30
gm_scopes(), 7, 30
gm_send_draft, 13, 16, 31
gm_send_message, 5, 14, 19, 20, 22, 23, 26, 29, 30, 32, 38, 39
gm_subject (gm_to), 34
gm_subject.mime (gm_mime), 24
gm_text_body (gm_mime), 24
gm_thread, 15, 27, 33, 34, 38, 40
gm_threads, 15, 27, 33, 33, 38, 40
gm_to, 34
gm_to.mime (gm_mime), 24
gm_token, 17, 35
gm_token_read (gm_token_write), 36
gm_token_write, 36
gm_trash_message, 5, 14, 19, 20, 22, 23, 26, 29, 30, 32, 37, 39
gm_trash_thread, 15, 27, 33, 34, 38, 40
gm_trash_thread(), 39
gm_untrash_message, 5, 14, 19, 20, 22, 23, 26, 29, 30, 32, 38, 39
gm_untrash_thread, 15, 27, 33, 34, 38, 39
gm_untrash_thread(), 38
gm_update_label, 12, 14, 21, 40
gm_update_label_patch
  (gm_update_label), 40
gmailr-configuration, 3
httr, 35
httr::config(), 7, 35, 37
jsonlite::fromJSON(), 6, 9
options and associated accessor functions, 4
quoted_printable_encode, 41
Token2.0, 7, 37
```