# Package ‘googleCloudStorageR’

January 5, 2021

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0.6.0</td>
</tr>
<tr>
<td>Title</td>
<td>Interface with Google Cloud Storage API</td>
</tr>
<tr>
<td>URL</td>
<td><a href="https://code.markedmondson.me/googleCloudStorageR/">https://code.markedmondson.me/googleCloudStorageR/</a></td>
</tr>
<tr>
<td>BugReports</td>
<td><a href="https://github.com/cloudyr/googleCloudStorageR/issues">https://github.com/cloudyr/googleCloudStorageR/issues</a></td>
</tr>
<tr>
<td>Depends</td>
<td>R (&gt;= 3.2.0)</td>
</tr>
<tr>
<td>Imports</td>
<td>assertthat (&gt;= 0.2.0), curl, googleAuthR (&gt;= 1.3.1), http (&gt;= 1.2.1), jsonlite (&gt;= 1.0), openssl, utils, yaml, zip (&gt;= 2.0.3)</td>
</tr>
<tr>
<td>Suggests</td>
<td>cli, fs, googleComputeEngineR, knitr, readr, rmarkdown, sodium, testthat, usethis</td>
</tr>
<tr>
<td>License</td>
<td>MIT + file LICENSE</td>
</tr>
<tr>
<td>LazyData</td>
<td>true</td>
</tr>
<tr>
<td>VignetteBuilder</td>
<td>knitr</td>
</tr>
<tr>
<td>RoxygenNote</td>
<td>7.1.0</td>
</tr>
<tr>
<td>NeedsCompilation</td>
<td>no</td>
</tr>
<tr>
<td>Author</td>
<td>Mark Edmondson [aut, cre] (<a href="https://orcid.org/0000-0002-8434-3881">https://orcid.org/0000-0002-8434-3881</a>)</td>
</tr>
<tr>
<td>Maintainer</td>
<td>Mark Edmondson <a href="mailto:r@sunholo.com">r@sunholo.com</a></td>
</tr>
<tr>
<td>Repository</td>
<td>CRAN</td>
</tr>
<tr>
<td>Date/Publication</td>
<td>2021-01-05 21:50:08 UTC</td>
</tr>
</tbody>
</table>

### R topics documented:

- `gcs_auth` .................................................. 2
- `gcs_compose_objects` .................................... 4
- `gcs_copy_object` .......................................... 5
- `gcs_create_bucket` ........................................ 6
- `gcs_create_bucket_acl` .................................... 7
**gs_auth**

Authenticate with Google Cloud Storage API

**Description**

Authenticate with Google Cloud Storage API

**Usage**

```r
gs_auth(json_file = NULL, token = NULL, email = NULL)
```
gcs_auth

Arguments

- **json_file**: Authentication json file you have downloaded from your Google Project
- **token**: An existing auth token you may have by other means
- **email**: The email to default authenticate through

Details

The best way to authenticate is to use an environment argument pointing at your authentication file. Set the file location of your download Google Project JSON file in a `GCS_AUTH_FILE` argument. Then, when you load the library you should auto-authenticate. However, you can authenticate directly using this function pointing at your JSON auth file.

Examples

```r
## Not run:
library(googleCloudStorageR)
gcs_auth("location_of_json_file.json")

# to use your own Google Cloud Project credentials
# go to GCP console and download client credentials JSON
# ideally set this in .Renviron file, not here but just for demonstration
Sys.setenv("GAR_CLIENT_JSON" = "location/of/file.json")
library(googleCloudStorageR)
# should now be able to log in via your own GCP project

gcs_auth()

# reauthentication
# Once you have authenticated, set email to skip the interactive message
gcs_auth(email = "my@email.com")

# or leave unset to bring up menu on which email to auth with
gcs_auth()
# The googleCloudStorageR package is requesting access to your Google account.
# Select a pre-authorised account or enter '0' to obtain a new token.
# Press Esc/Ctrl + C to abort.
#1: my@email.com
#2: work@mybusiness.com
# you can set authentication for many emails, then switch between them e.g.
gcs_auth(email = "my@email.com")
gcs_list_buckets("my-project") # lists what buckets you have access to
gcs_auth(email = "work@mybusiness.com")
gcs_list_buckets("my-project") # lists second set of buckets

## End(Not run)
```
gcs-compose-objects

Description

This merges objects stored on Cloud Storage into one object.

Usage

gcs-compose-objects(objects, destination, bucket = gcs-get-global-bucket())

Arguments

objects A character vector of object names to combine
destination Name of the new object.
bucket The bucket where the objects sit

Value

Object metadata

See Also

Compose objects

Other object functions: gcs-copy-object(), gcs-delete-object(), gcs-get-object(), gcs-list-objects(), gcs-metadata-object()

Examples

## Not run:
gcs-global-bucket("your-bucket")
objs <- gcs-list-objects()

compose_me <- objs$name[1:30]

gcs-compose-objects(compose_me, "composed/test.json")

## End(Not run)
Copy an object

Description

Copies an object to a new destination

Usage

gcs_copy_object(
    source_object,
    destination_object,
    source_bucket = gcs_get_global_bucket(),
    destination_bucket = gcs_get_global_bucket(),
    rewriteToken = NULL,
    destinationPredefinedAcl = NULL
)

Arguments

source_object  The name of the object to copy, or a gs:// URL
destination_object  The name of where to copy the object to, or a gs:// URL
source_bucket  The bucket of the source object
destination_bucket  The bucket of the destination
rewriteToken  Include this field (from the previous rewrite response) on each rewrite request after the first one, until the rewrite response ‘done’ flag is true.
destinationPredefinedAcl  Apply a predefined set of access controls to the destination object. If not NULL must be one of the predefined access controls such as “bucketOwnerFullControl”

Value

If successful, a rewrite object.

See Also

Other object functions: gcs_compose_objects(), gcs_delete_object(), gcs_get_object(), gcs_list_objects(), gcs_metadata_object()
gcs_create_bucket  Create a new bucket

Description

Create a new bucket in your project

Usage

```r
gcs_create_bucket(
  name,
  projectId,
  location = "US",
  storageClass = c("MULTI_REGIONAL", "REGIONAL", "STANDARD", "NEARLINE", "COLDLINE",
  "DURABLE_REduced_Availability"),
  predefinedAcl = c("projectPrivate", "authenticatedRead", "private", "publicRead",
  "publicReadWrite"),
  predefinedDefaultObjectAcl = c("bucketOwnerFullControl", "bucketOwnerRead",
  "authenticatedRead", "private", "projectPrivate", "publicRead"),
  projection = c("noAcl", "full"),
  versioning = FALSE,
  lifecycle = NULL
)
```

Arguments

- **name**: Globally unique name of bucket to create
- **projectId**: A valid Google project id
- **location**: Location of bucket. See details
- **storageClass**: Type of bucket
- **predefinedAcl**: Apply predefined access controls to bucket
- **predefinedDefaultObjectAcl**: Apply predefined access controls to objects
- **projection**: Properties to return. Default noAcl omits acl properties
- **versioning**: Set if the bucket supports versioning of its objects
- **lifecycle**: A list of `gcs_create_lifecycle` objects

Details

See here for details on location options

See Also

Other bucket functions: `gcs_create_lifecycle()`, `gcs_delete_bucket()`, `gcs_get_bucket()`, `gcs_get_global_bucket()`, `gcs_global_bucket()`, `gcs_list_buckets()`
gcs_create_bucket_acl  Create a Bucket Access Controls

Description

Create a new access control at the bucket level

Usage

gcs_create_bucket_acl(
  bucket = gcs_get_global_bucket(),
  entity = "",
  entity_type = c("user", "group", "domain", "project", "allUsers",
                   "allAuthenticatedUsers"),
  role = c("READER", "OWNER")
)

Arguments

bucket  Name of a bucket, or a bucket object returned by gcs_create_bucket
entity   The entity holding the permission. Not needed for entity_type allUsers or allAuthenticatedUsers
entity_type   what type of entity
role   Access permission for entity
       Used also for when a bucket is updated

Value

Bucket access control object

See Also

Other Access control functions: gcs_get_bucket_acl(), gcs_get_object_acl(), gcs_update_object_acl()

gcs_create_lifecycle  Create a lifecycle condition

Description

Use this to set rules for how long objects last in a bucket in gcs_create_bucket
Usage

gcs_create_lifecycle(
    age = NULL,
    createdBefore = NULL,
    numNewerVersions = NULL,
    isLive = NULL
)

Arguments

age            Age in days before objects are deleted
createdBefore  Deletes all objects before this date
numNewerVersions Deletes all newer versions of this object
isLive         If TRUE deletes all live objects, if FALSE deletes all archived versions

numNewerVersions and isLive works only for buckets with object versioning
For multiple conditions, pass this object in as a list.

See Also

Lifecycle documentation https://cloud.google.com/storage/docs/lifecycle

Other bucket functions: gcs_create_bucket(), gcs_delete_bucket(), gcs_get_bucket(), gcs_get_global_bucket(),
gcs_global_bucket(), gcs_list_buckets()

Examples

## Not run:
lifecycle <- gcs_create_lifecycle(age = 30)

    gcs_create_bucket("your-bucket-lifecycle",
        projectId = "your-project",
        location = "EUROPE-NORTH1",
        storageClass = "REGIONAL",
        lifecycle = list(lifecycle))

## End(Not run)

---

## gcs_create_pubsub

Create a pub/sub notification for a bucket

Description

Add a notification configuration that sends notifications for all supported events.
Usage

gcs_create_pubsub(
  topic,
  project,
  bucket = gcs_get_global_bucket(),
  event_types = NULL
)

Arguments

topic The pub/sub topic name
project The project-id that has the pub/sub topic
bucket The bucket for notifications
event_types What events to activate, leave at default for all

Details

Cloud Pub/Sub notifications allow you to track changes to your Cloud Storage objects. As a minimum you will need: the Cloud Pub/Sub API activated for the project; sufficient permissions on the bucket you wish to monitor; sufficient permissions on the project to receive notifications; an existing pub/sub topic; have given your service account at least pubsub.publisher permission.

See Also

https://cloud.google.com/storage/docs/reporting-changes

Other pubsub functions: gcs_delete_pubsub(), gcs_get_service_email(), gcs_list_pubsub()

Examples

## Not run:

project <- "myproject"
bucket <- "mybucket"

# get the email to give access
gcs_get_service_email(project)

# once email has access, create a new pub/sub topic for your bucket
gcs_create_pubsub("gcs_r", project, bucket)

## End(Not run)
### gcs_delete_bucket

Delete a bucket

**Description**

Delete the bucket, and all its objects

**Usage**

```python
gcs_delete_bucket(
    bucket,
    ifMetagenerationMatch = NULL,
    ifMetagenerationNotMatch = NULL
)
```

**Arguments**

- `bucket`: Name of the bucket, or a bucket object
- `ifMetagenerationMatch`: Delete only if metageneration matches
- `ifMetagenerationNotMatch`: Delete only if metageneration does not match

**See Also**

Other bucket functions: `gcs_create_bucket()`, `gcs_create_lifecycle()`, `gcs_get_bucket()`, `gcs_get_global_bucket()`, `gcs_global_bucket()`, `gcs_list_buckets()`

---

### gcs_delete_object

Delete an object

**Description**

Deletes an object from a bucket

**Usage**

```python
gcs_delete_object(
    object_name,
    bucket = gcs_get_global_bucket(),
    generation = NULL
)
```
**Arguments**

- **object_name**: Object to be deleted, or a gs:// URL
- **bucket**: Bucket to delete object from
- **generation**: If present, deletes a specific version.
  Default if generation is NULL is to delete the latest version.

**Value**

If successful, TRUE.

**See Also**

Other object functions: `gcs_compose_objects()`, `gcs_copy_object()`, `gcs_get_object()`, `gcs_list_objects()`, `gcs_metadata_object()`

---

**gcs_delete_pubsub**  
Delete pub/sub notifications for a bucket

**Description**

Delete notification configurations for a bucket.

**Usage**

```
gcs_delete_pubsub(config_name, bucket = gcs_get_global_bucket())
```

**Arguments**

- **config_name**: The ID of the pubsub configuration
- **bucket**: The bucket for notifications

**Details**

Cloud Pub/Sub notifications allow you to track changes to your Cloud Storage objects. As a minimum you will need: the Cloud Pub/Sub API activated for the project; sufficient permissions on the bucket you wish to monitor; sufficient permissions on the project to receive notifications; an existing pub/sub topic; have given your service account at least pubsub.publisher permission.

**Value**

TRUE if successful

**See Also**

- [https://cloud.google.com/storage/docs/reporting-changes](https://cloud.google.com/storage/docs/reporting-changes)
- Other pubsub functions: `gcs_create_pubsub()`, `gcs_get_service_email()`, `gcs_list_pubsub()`
gcs_download_url  

*Get the download URL*

**Description**

Create the download URL for objects in buckets

**Usage**

```r
  gcs_download_url(object_name, bucket = gcs_get_global_bucket(), public = FALSE)
```

**Arguments**

- `object_name`  
  A vector of object names

- `bucket`  
  A vector of bucket names

- `public`  
  TRUE to return a public URL

**Details**

- bucket names should be length 1 or same length as `object_name`

- Download URLs can be either authenticated behind a login that you may need to update access for via `gcs_update_object_acl`, or public to all if their `predefinedAcl = 'publicRead'`

- Use the `public = TRUE` to return the URL accessible to all, which changes the domain name from `storage.cloud.google.com` to `storage.googleapis.com`

**Value**

the URL for downloading objects

**See Also**

- Other download functions: `gcs_parse_download()`, `gcs_signed_url()`

---

gcs_first  

*Save your R session to the cloud on startup/exit*

**Description**

Place within your `.Rprofile` to load and save your session data automatically

**Usage**

```r
  gcs_first(bucket = Sys.getenv("GCS_SESSION_BUCKET"))
  gcs_last(bucket = Sys.getenv("GCS_SESSION_BUCKET"))
```
Arguments

bucket The bucket holding your session data. See Details.

Details

The folder you want to save to Google Cloud Storage will also need to have a yaml file called _gcssave.yaml in the root of the directory. It can hold the following arguments:

- [Required] bucket - the GCS bucket to save to
- [Optional] loaddir - if the folder name is different to the current, where to load the R session from
- [Optional] pattern - a regex of what files to save at the end of the session
- [Optional] load_on_startup - if FALSE will not attempt to load on startup

The bucket name is also set via the environment arg GCE_SESSION_BUCKET. The yaml bucket name will take precedence if both are set.

The folder is named on GCS the full working path to the working directory e.g. /Users/mark/dev/your-r-project which is what is looked for on startup. If you create a new R project with the same filepath and bucket as an existing saved set, the files will download automatically when you load R from that folder (when starting an RStudio project).

If you load from a different filepath (e.g. with loaddir set in yaml), when you exit and save the files will be saved under your new present working directory.

Files with the same name will not be overwritten. If you want them to be, delete or rename them then reload the R session.

This function does not act like git, or intended as a replacement - its main use is imagined to be for using RStudio Server within disposable Docker containers on Google Cloud Engine (e.g. via googleComputeEngineR)

For authentication for GCS, the easiest way is to make sure your authentication file is available in environment file GCS_AUTH_FILE, or if on Google Compute Engine it will reuse the Google Cloud authentication via gar_gce_auth

See Also
gcs_save_all and gcs_load_all that these functions call

Examples

```r
## Not run:
.First <- function(){
  googleCloudStorageR::gcs_first()
}

.Last <- function(){
```
### gcs_get_bucket

#### Get bucket info

**Description**

Meta data about the bucket

**Usage**

```r

# Set bucket
bucket = gcs_get_global_bucket()
ifMetagenerationMatch = NULL,
ifMetagenerationNotMatch = NULL,
projection = c("noAcl", "full")
```

**Arguments**

- `bucket`: Name of a bucket, or a bucket object returned by `gcs_create_bucket`
- `ifMetagenerationMatch`: Return only if metageneration matches
- `ifMetagenerationNotMatch`: Return only if metageneration does not match
- `projection`: Properties to return. Default noAcl omits acl properties

**Value**

A bucket resource object

**See Also**

Other bucket functions: `gcs_create_bucket()`, `gcs_create_lifecycle()`, `gcs_delete_bucket()`, `gcs_get_global_bucket()`, `gcs_global_bucket()`, `gcs_list_buckets()`
### Examples

```r
## Not run:
buckets <- gcs_list_buckets("your-project")
## use the name of the bucket to get more meta data
bucket_meta <- gcs_get_bucket(buckets$name[[1]])

## End(Not run)
```

---

**gcs_get_bucket_acl**  
*Get Bucket Access Controls*

**Description**

Returns the ACL entry for the specified entity on the specified bucket

**Usage**

```r
gcs_get_bucket_acl(
  bucket = gcs_get_global_bucket(),
  entity = "",
  entity_type = c("user", "group", "domain", "project", "allUsers",
                  "allAuthenticatedUsers")
)
```

**Arguments**

- **bucket**  
  Name of a bucket, or a bucket object returned by `gcs_create_bucket`

- **entity**  
  The entity holding the permission. Not needed for `entity_type allUsers` or `allAuthenticatedUsers`

- **entity_type**  
  what type of entity  
  Used also for when a bucket is updated

**Value**

Bucket access control object

**See Also**

Other Access control functions: `gcs_create_bucket_acl()`, `gcs_get_object_acl()`, `gcs_update_object_acl()`
Examples

```r
## Not run:

buck_meta <- gcs_get_bucket(projection = "full")

acl <- gcs_get_bucket_acl(entity_type = "project",
                           entity = gsub("project-", "",
                                         buck_meta$acl$entity[[1]])

## End(Not run)
```

---

**gcs_get_global_bucket**  
*Get global bucket name*

---

Description

Bucket name set this session to use by default

Usage

```r
gcs_get_global_bucket()
```

Details

Set the bucket name via `gcs_global_bucket`

Value

Bucket name

See Also

Other bucket functions: `gcs_create_bucket()`, `gcs_create_lifecycle()`, `gcs_delete_bucket()`, `gcs_get_bucket()`, `gcs_global_bucket()`, `gcs_list_buckets()`
**gcs_get_object**

Get an object in a bucket directly

**Description**

This retrieves an object directly.

**Usage**

```r

gcs_get_object(
  object_name,
  bucket = gcs_get_global_bucket(),
  meta = FALSE,
  saveToDisk = NULL,
  overwrite = FALSE,
  parseObject = TRUE,
  parseFunction = gcs_parse_download
)
```

**Arguments**

- **object_name**: name of object in the bucket that will be URL encoded, or a gs:// URL
- **bucket**: bucket containing the objects. Not needed if using a gs:// URL
- **meta**: If TRUE then get info about the object, not the object itself
- **saveToDisk**: Specify a filename to save directly to disk
- **overwrite**: If saving to a file, whether to overwrite it
- **parseObject**: If saveToDisk is NULL, whether to parse with parseFunction
- **parseFunction**: If saveToDisk is NULL, the function that will parse the download. Defaults to gcs_parse_download

**Details**

This differs from providing downloads via a download link as you can do via gcs_download_url. object_name can use a gs:// URI instead, in which case it will take the bucket name from that URI and bucket argument will be overridden. The URLs should be in the form gs://bucket/object/name.

By default if you want to get the object straight into an R session the parseFunction is gcs_parse_download which wraps httr’s content.

If you want to use your own function (say to unzip the object) then supply it here. The first argument should take the downloaded object.

**Value**

The object, or TRUE if successfully saved to disk.
See Also

Other object functions: `gcs_compose_objects()`, `gcs_copy_object()`, `gcs_delete_object()`, `gcs_list_objects()`, `gcs_metadata_object()`

Examples

```r
## Not run:

## something to download
## data.frame that defaults to be called "mtcars.csv"
gcs_upload(mtcars)

## get the mtcars csv from GCS, convert it to an R obj
gcs_get_object("mtcars.csv")

## get the mtcars csv from GCS, save it to disk
gcs_get_object("mtcars.csv", saveToDisk = "mtcars.csv")

## default gives a warning about missing column name.
## custom parse function to suppress warning
f <- function(object){
  suppressWarnings(httr::content(object, encoding = "UTF-8"))
}

## get mtcars csv with custom parse function.
gcs_get_object("mtcars_meta.csv", parseFunction = f)

## End(Not run)
```

---

**gcs_get_object_acl**

Check the access control settings for an object for one entity

Description

Returns the default object ACL entry for the specified entity on the specified bucket.

Usage

```r
gcs_get_object_acl(
  object_name,
  bucket = gcs_get_global_bucket(),
  entity = "",
  entity_type = c("user", "group", "domain", "project", "allUsers", "allAuthenticatedUsers"),
  generation = NULL
)
```
Arguments

- **object_name**: Name of the object
- **bucket**: Name of a bucket
- **entity**: The entity holding the permission. Not needed for entity_type allUsers or allAuthenticatedUsers
- **entity_type**: The type of entity
- **generation**: If present, selects a specific revision of the object

See Also

Other Access control functions: `gcs_create_bucket_acl()`, `gcs_get_bucket_acl()`, `gcs_update_object_acl()`

Examples

```r
## Not run:

# single user

# single user

gcs_update_object_acl("mtcars.csv",
bucket = gcs_get_global_bucket(),
entity = "joe@blogs.com",
entity_type = "user")

acl <- gcs_get_object_acl("mtcars.csv", entity = "joe@blogs.com")

# all users

gcs_update_object_acl("mtcars.csv",
bucket = gcs_get_global_bucket(),
entity_type = "allUsers")

acl <- gcs_get_object_acl("mtcars.csv", entity_type = "allUsers")

## End(Not run)
```

---

**gcs_get_service_email**  
*Get the email of service account associated with the bucket*

Description

Use this to get the right email so you can give it pubsub.publisher permission.

Usage

`gcs_get_service_email(project)`
Arguments

project  The project name containing the bucket

Details

This service email can be different from the email in the service JSON. Give this pubsub.publisher permission in the Google cloud console.

See Also

Other pubsub functions: \texttt{gcs\_create\_pubsub()}, \texttt{gcs\_delete\_pubsub()}, \texttt{gcs\_list\_pubsub()}

\begin{verbatim}
gcs\_global\_bucket bucket

gcs\_global\_bucket(bucket)
\end{verbatim}

Description

Set a bucket name used for this R session

Usage

\begin{verbatim}
gcs\_global\_bucket(bucket)
\end{verbatim}

Arguments

bucket  bucket name you want this session to use by default, or a bucket object

Details

This sets a bucket to a global environment value so you don’t need to supply the bucket argument to other API calls.

Value

The bucket name (invisibly)

See Also

Other bucket functions: \texttt{gcs\_create\_bucket()}, \texttt{gcs\_create\_lifecycle()}, \texttt{gcs\_delete\_bucket()}, \texttt{gcs\_get\_bucket()}, \texttt{gcs\_get\_global\_bucket()}, \texttt{gcs\_list\_buckets()}
**gcs_list_buckets**

**List buckets**

Description

List the buckets your projectId has access to

Usage

```r
gcs_list_buckets(
  projectId,
  prefix = "",
  projection = c("noAcl", "full"),
  maxResults = 1000,
  detail = c("summary", "full")
)
```

Arguments

- `projectId`: Project containing buckets to list
- `prefix`: Filter results to names beginning with this prefix
- `projection`: Properties to return. Default noAcl omits acl properties
- `maxResults`: Max number of results
- `detail`: Set level of detail

Details

Columns returned by `detail` are:

- `summary` - name, storageClass, location, updated
- `full` - as above plus: id, selfLink, projectNumber, timeCreated, metageneration, etag

Value

`data.frame` of buckets

See Also

Other bucket functions: `gcs_create_bucket()`, `gcs_create_lifecycle()`, `gcs_delete_bucket()`, `gcs_get_bucket()`, `gcs_get_global_bucket()`, `gcs_global_bucket()`
Examples

## Not run:

buckets <- gcs_list_buckets("your-project")

## use the name of the bucket to get more meta data
bucket_meta <- gcs_get_bucket(buckets$name[[1]])

## End(Not run)

---

gcs_list_objects List objects in a bucket

Description

List objects in a bucket

Usage

gcs_list_objects(
  bucket = gcs_get_global_bucket(),
  detail = c("summary", "more", "full"),
  prefix = NULL,
  delimiter = NULL
)

Arguments

- **bucket**: bucket containing the objects
- **detail**: Set level of detail
- **prefix**: Filter results to objects whose names begin with this prefix
- **delimiter**: Use to list objects like a directory listing.

Details

Columns returned by `detail` are:

- **summary** - name, size, updated
- **more** - as above plus: bucket, contentType, storageClass, timeCreated
- **full** - as above plus: id, selfLink, generation, metageneration, md5Hash, mediaLink, crc32c, etag
delimited returns results in a directory-like mode: items will contain only objects whose names, aside from the prefix, do not contain delimiter. In conjunction with the prefix filter, the use of the delimiter parameter allows the list method to operate like a directory listing, despite the object namespace being flat. For example, if delimiter were set to "/", then listing objects from a bucket that contains the objects "a/b","a/c","dddd","eeee","e/f" would return objects "dddd" and "eeee", and prefixes "a/" and "e/".

Value
A data.frame of the objects

See Also
Other object functions: `gcs_compose_objects()`, `gcs_copy_object()`, `gcs_delete_object()`, `gcs_get_object()`, `gcs_metadata_object()`
gcs_load

Load .RData objects or sessions from the Google Cloud

Description

Load R objects that have been saved using gcs_save or gcs_save_image

Usage

```r
gcs_load(
  file = ".RData",
  bucket = gcs_get_global_bucket(),
  envir = .GlobalEnv,
  saveToDisk = file,
  overwrite = TRUE
)
```

Arguments

- **file**: Where the files are stored
- **bucket**: Bucket the stored objects are in
- **envir**: Environment to load objects into
- **saveToDisk**: Where to save the loaded file. Default same file name
- **overwrite**: If file exists, overwrite. Default TRUE.

Details

The argument file’s default is to load an image file called .RData from gcs_save_image into the Global environment.

This would overwrite your existing .RData file in the working directory, so change the file name if you don’t wish this to be the case.

Value

TRUE if successful

See Also

Other R session data functions: gcs_save_all(), gcs_save_image(), gcs_save(), gcs_source()
`gcs_metadata_object`  Make metadata for an object

**Description**

Use this to pass to uploads in `gcs_upload`.

**Usage**

```r
gcs_metadata_object(
  object_name = NULL,
  metadata = NULL,
  md5Hash = NULL,
  crc32c = NULL,
  contentLanguage = NULL,
  contentEncoding = NULL,
  contentDisposition = NULL,
  cacheControl = NULL
)
```

**Arguments**

- `object_name` Name of the object. GCS uses this version if also set elsewhere, or a gs:// URL.
- `metadata` User-provided metadata, in key/value pairs.
- `md5Hash` MD5 hash of the data; encoded using base64.
- `crc32c` CRC32c checksum, as described in RFC 4960, Appendix B; encoded using base64 in big-endian byte order.
- `contentLanguage` Content-Language of the object data.
- `contentEncoding` Content-Encoding of the object data.
- `contentDisposition` Content-Disposition of the object data.
- `cacheControl` Cache-Control directive for the object data.

**Value**

Object metadata for uploading of class `gar_Object`.

**See Also**

Other object functions: `gcs_compose_objects()`, `gcs_copy_object()`, `gcs_delete_object()`, `gcs_get_object()`, `gcs_list_objects()`
gcs_parse_download  Parse downloaded objects straight into R

Description
Wrapper for httr's content. This is the default function used in gcs_get_object

Usage
  gcs_parse_download(object, encoding = "UTF-8")

Arguments
  object  The object downloaded
  encoding  Default to UTF-8

See Also
  gcs_get_object
  Other download functions: gcs_download_url(), gcs_signed_url()

---

gcs_retry_upload  Retry a resumeable upload

Description
Used internally in gcs_upload, you can also use this for failed uploads within one week of generating the upload URL

Usage
  gcs_retry_upload(
    retry_object = NULL,
    upload_url = NULL,
    file = NULL,
    type = NULL
  )

Arguments
  retry_object  A object of class gcs_upload_retry.
  upload_url  As created in a failed upload via gcs_upload
  file  The file location to upload
  type  The file type, guessed if NULL
  Either supply a retry object, or the upload_url, file and type manually yourself. The function will first check to see how much has been uploaded already, then try to send up the remaining bytes.
**gcs_save**

Save .RData objects to the Google Cloud

**Description**

Performs save then saves it to Google Cloud Storage.

**Usage**

```r
gcs_save(..., file, bucket = gcs_get_global_bucket(), envir = parent.frame())
```

**Arguments**

- `...` The names of the objects to be saved (as symbols or character strings).
- `file` The file name that will be uploaded (conventionally with file extension .RData)
- `bucket` Bucket to store objects in
- `envir` Environment to search for objects to be saved

**Details**

For all session data use `gcs_save_image` instead.

```r
gcs_save(ob1, ob2, ob3, file = "mydata.RData")
```
will save the objects specified to an .RData file then save it to Cloud Storage, to be loaded later using `gcs_load`.

For any other use, its better to use `gcs_upload` and `gcs_get_object` instead.

Restore the R objects using `gcs_load(bucket = "your_bucket")`

This will overwrite any data within your local environment with the same name.

**Value**

The GCS object

**See Also**

Other R session data functions: `gcs_load()`, `gcs_save_all()`, `gcs_save_image()`, `gcs_source()`
gcs_save_all

Save/Load all files in directory to Google Cloud Storage

Description

This function takes all the files in the directory, zips them, and saves/loads/deletes them to the cloud. The upload name will be the directory name.

Usage

```r
gcs_save_all(
directory = getwd(),
bucket = gcs_get_global_bucket(),
pattern = "",
predefinedAcl = c("private", "bucketLevel", "authenticatedRead",
  "bucketOwnerFullControl", "bucketOwnerRead", "projectPrivate", "publicRead",
  "default")
)

gcs_load_all(
directory = getwd(),
bucket = gcs_get_global_bucket(),
exdir = directory,
list = FALSE
)

gcs_delete_all(directory = getwd(), bucket = gcs_get_global_bucket())
```

Arguments

- **directory**: The folder to upload/download
- **bucket**: Bucket to store within
- **pattern**: An optional regular expression. Only file names which match the regular expression will be saved.
- **predefinedAcl**: Specify user access to object. Default is 'private'. Set to 'bucketLevel' for buckets with bucket level access enabled.
- **exdir**: When downloading, specify a destination directory if required
- **list**: When downloading, only list where the files would unzip to

Details

Zip/unzip is performed before upload and after download using `zip`.

Value

When uploading the GCS meta object; when downloading `TRUE` if successful


See Also

Other R session data functions: `gcs_load()`, `gcs_save_image()`, `gcs_save()`, `gcs_source()`

Examples

```r
## Not run:

gcs_save_all(
    directory = "path-to-all-images",
    bucket = "my-bucket",
    predefinedAcl = "bucketLevel")

## End(Not run)
```

```
gcs_save_image

Save an R session to the Google Cloud

Description

Performs `save.image` then saves it to Google Cloud Storage.

Usage

```r
gcs_save_image(
    file = ".RData",
    bucket = gcs_get_global_bucket(),
    saveLocation = NULL,
    envir = parent.frame()
)
```

Arguments

- `file`: Where to save the file in GCS and locally
- `bucket`: Bucket to store objects in
- `saveLocation`: Which folder in the bucket to save file
- `envir`: Environment to save from

Details

- `gcs_save_image(bucket = "your_bucket")` will save all objects in the workspace to `.RData` folder on Google Cloud Storage within `your_bucket`.
- Restore the objects using `gcs_load(bucket = "your_bucket")`
- This will overwrite any data with the same name in your current local environment.
**gcs_setup**

Value

The GCS object

See Also

Other R session data functions: `gcs_load()`, `gcs_save_all()`, `gcs_save()`, `gcs_source()`

---

**gcs_setup**  
*Help set-up googleCloudStorageR*

Description

Use this to make a wizard to walk through set-up steps

Usage

```r
gcs_setup()
```

Details

This function assumes you have at least a Google Cloud Platform project setup, from which it can generate the necessary authentication keys and set up authentication.

It uses `gar_setup_menu` to create the wizard. You will need to have owner access to the project you are using.

After each menu option has completed, restart R and reload the library and this function to continue to the next step.

Upon successful set-up, you should see a message similar to `Successfully auto-authenticated via /xxxx/googlecloudstorager-auth-key.json` and `Set default bucket name to 'xxxx' when you load the library via `library(googleCloudStorageR)``

See Also

Setup documentation on googleCloudStorageR website

Examples

```r
## Not run:
library(googleCloudStorageR)
gcs_setup()

## End(Not run)
```
Description

This creates a signed URL which you can share with others who may or may not have a Google account. The object will be available until the specified timestamp.

Usage

```r

gcs_signed_url(
  meta_obj,
  expiration_ts = Sys.time() + 3600,
  verb = "GET",
  md5hash = NULL,
  includeContentType = FALSE
)
```

Arguments

- `meta_obj`: A meta object from `gcs_get_object`
- `expiration_ts`: A timestamp of class "POSIXct" such as from `Sys.time()` or a numeric in seconds from Unix Epoch. Default is 60 mins.
- `verb`: The URL verb of access e.g. `GET` or `PUT`. Default `GET`
- `md5hash`: An optional md5 digest value
- `includeContentType`: For getting the URL via browsers this should be set to `FALSE` (the default). Otherwise, set to `TRUE` to include the content type of the object in the request needed.

Details

Create a URL with a time-limited read and write to an object, regardless whether they have a Google account

See Also

- `https://cloud.google.com/storage/docs/access-control/signed-urls`
- Other download functions: `gcs_download_url()`, `gcs_parse_download()`

Examples

```r
## Not run:

obj <- gcs_get_object("your_file", meta = TRUE)

signed <- gcs_signed_url(obj)
```
temp <- tempfile()
on.exit(unlink(temp))
download.file(signed, destfile = temp)
file.exists(temp)

## End(Not run)

gcs_source

Source an R script from the Google Cloud

Description

Download an R script and run it immediately via `source`

Usage

gcs_source(script, bucket = gcs_get_global_bucket(), ...)

Arguments

- `script` The name of the script on GCS
- `bucket` Bucket the stored objects are in
- `...` Passed to `source`

Value

TRUE if successful

See Also

Other R session data functions: `gcs_load()`, `gcs_save_all()`, `gcs_save_image()`, `gcs_save()`
Description

Updates Google Cloud Storage ObjectAccessControls

Usage

gcs_update_object_acl(
  object_name,
  bucket = gcs_get_global_bucket(),
  entity = "",
  entity_type = c("user", "group", "domain", "project", "allUsers",
  "allAuthenticatedUsers"),
  role = c("READER", "OWNER")
)

Arguments

object_name    Object to update
bucket         Google Cloud Storage bucket
entity         entity to update or add, such as an email
entity_type    what type of entity
role           Access permission for entity

Details

An entity is an identifier for the entity_type.

- entity="user" may have userId or email
- entity="group" may have groupId or email
- entity="domain" may have domain
- entity="project" may have team-projectId

For example:

- entity="user" could be jane@doe.com
- entity="group" could be example@googlegroups.com
- entity="domain" could be example.com which is a Google Apps for Business domain.

Value

TRUE if successful
**gcs_upload**

Upload a file of arbitrary type

**Description**

Upload up to 5TB

**Usage**

```r
# gcs_upload
file, bucket = gcs_get_global_bucket(),
type = NULL,
name = deparse(substitute(file)),
object_function = NULL,
object_metadata = NULL,
predefinedAcl = c("private", "bucketLevel", "authenticatedRead",
    "bucketOwnerFullControl", "bucketOwnerRead", "projectPrivate", "publicRead",
    "default"),
upload_type = c("simple", "resumable")

# gcs_upload_set_limit
```

**Arguments**

- `file`: data.frame, list, R object or filepath (character) to upload file
- `bucket`: bucketname you are uploading to
- `type`: MIME type, guessed from file extension if NULL
- `name`: What to call the file once uploaded. Default is the filepath
- `object_function`: If not NULL, a function(input,output)
- `object_metadata`: Optional metadata for object created via `gcs_metadata_object`
- `predefinedAcl`: Specify user access to object. Default is 'private'. Set to 'bucketLevel' for buckets with bucket level access enabled.
- `upload_type`: Override automatic decision on upload type
- `upload_limit`: Upload limit in bytes

---

**See Also**

- `objectAccessControls` on Google API reference
- Other Access control functions: `gcs_create_bucket_acl`, `gcs_get_bucket_acl`, `gcs_get_object_acl`
Details

When using object_function it expects a function with two arguments:

- **input** The object you supply in file to write from
- **output** The filename you write to

By default the upload_type will be 'simple' if under 5MB, 'resumable' if over 5MB. Use gcs_upload_set_limit to modify this boundary - you may want it smaller on slow connections, higher on faster connections. 'Multipart' upload is used if you provide a object_metadata.

If object_function is NULL and file is not a character filepath, the defaults are:

- file's class is data.frame - write.csv
- file's class is list - toJSON

If object_function is not NULL and file is not a character filepath, then object_function will be applied to the R object specified in file before upload. You may want to also use name to ensure the correct file extension is used e.g. name = 'myobject.feather'

If file or name argument contains folders e.g. /data/file.csv then the file will be uploaded with the same folder structure e.g. in a /data/ folder. Use name to override this.

Value

If successful, a metadata object

scopes

Requires scopes https://www.googleapis.com/auth/devstorage.read_write or https://www.googleapis.com/auth/devstorage.full_control

Examples

```
## Not run:
## set global bucket so don't need to keep supplying in future calls
gcs_global_bucket("my-bucket")

## by default will convert dataframes to csv
gcs_upload(mtcars)

## mtcars has been renamed to mtcars.csv
gcs_list_objects()

## to specify the name, use the name argument
gcs_upload(mtcars, name = "my_mtcars.csv")

## when looping, its best to specify the name else it will take
## the deparsed function call e.g. X[[i]]
my_files <- list.files("my_uploads")
lapply(my_files, function(x) gcs_upload(x, name = x))
```
## you can supply your own function to transform R objects before upload

```r
f <- function(input, output){
  write.csv2(input, file = output)
}
```

gcs_upload(mtcars, name = "mtcars_csv2.csv", object_function = f)

# upload to a bucket with bucket level ACL set

gcs_upload(mtcars, predefinedAcl = "bucketLevel")

# modify boundary between simple and resumable uploads
# default 5000000L is 5MB

gcs_upload_set_limit(1000000L)

## End(Not run)

---

`gcs_version_bucket` *Change or fetch bucket version status*

### Description

Turn bucket versioning on or off, check status (default), or list archived versions of objects in the bucket and view their generation numbers.

### Usage

```r
gcs_version_bucket(bucket, action = c("status", "enable", "disable", "list"))
```

### Arguments

- `bucket` gcs bucket
- `action` "status", "enable", "disable", or "list"

### Value

- If `action="list"` a versioned_objects dataframe
- If `action="status"` a boolean on if versioning is TRUE or FALSE
- If `action="enable"` or "disable" TRUE if operation is successful

### Examples

```r
## Not run:
  buck <- gcs_get_global_bucket()
  gcs_version_bucket(buck, action = "disable")
  gcs_version_bucket(buck, action = "status")
```
# Versioning is NOT ENABLED for "your-bucket"
gcs_version_bucket(buck, action = "enable")
# TRUE
gcs_version_bucket(buck, action = "status")
# Versioning is ENABLED for "your-bucket"
gcs_version_bucket(buck, action = "list")

## End(Not run)

---

**Description**

Interact with Google Cloud Storage API in R. Part of the 'cloudyr' project.
Index

* Access control functions
  gcs_create_bucket_acl, 7
  gcs_get_bucket_acl, 15
  gcs_get_object_acl, 18
  gcs_update_object_acl, 33

* R session data functions
  gcs_load, 24
  gcs_save, 27
  gcs_save_all, 28
  gcs_save_image, 29
  gcs_source, 32

* bucket functions
  gcs_create_bucket, 6
  gcs_create_lifecycle, 7
  gcs_delete_bucket, 10
  gcs_get_bucket, 14
  gcs_get_global_bucket, 16
  gcs_global_bucket, 20
  gcs_list_buckets, 21

* download functions
  gcs_download_url, 12
  gcs_parse_download, 26
  gcs_signed_url, 31

* object functions
  gcs_compose_objects, 4
  gcs_copy_object, 5
  gcs_delete_object, 10
  gcs_get_object, 17
  gcs_list_objects, 22
  gcs_metadata_object, 25

* pubsub functions
  gcs_create_pubsub, 8
  gcs_delete_pubsub, 11
  gcs_get_service_email, 19
  gcs_list_pubsub, 23

content, 17, 26

gar_gce_auth, 13
gar_setup_menu, 30

gcs_auth, 2
gcs_compose_objects, 4, 5, 11, 18, 23, 25
gcs_copy_object, 4, 5, 11, 18, 23, 25
gcs_create_bucket, 6, 7, 8, 10, 14–16, 20, 21
gcs_create_bucket_acl, 7, 15, 19, 34
gcs_create_lifecycle, 6, 7, 10, 14, 16, 20, 21
gcs_create_pubsub, 8, 11, 20, 23
gcs_delete_all (gcs_save_all), 28
gcs_delete_bucket, 6, 8, 10, 14, 16, 20, 21
gcs_delete_object, 4, 5, 10, 18, 23, 25
gcs_delete_pubsub, 9, 11, 20, 23
gcs_download_url, 12, 17, 26, 31
gcs_first, 12
gcs_get_bucket, 6, 8, 10, 14, 16, 20, 21
gcs_get_bucket_acl, 7, 15, 19, 34
gcs_get_global_bucket, 6, 8, 10, 14, 16, 20, 21
gcs_get_object, 4, 5, 11, 17, 23, 25–27, 31
gcs_get_object_acl, 7, 15, 18, 34
gcs_get_service_email, 9, 11, 19, 23
gcs_global_bucket, 6, 8, 10, 14, 16, 20, 21
gcs_last (gcs_first), 12
gcs_list_buckets, 6, 8, 10, 14, 16, 20, 21
gcs_list_objects, 4, 5, 11, 18, 22, 25
gcs_list_pubsub, 9, 11, 20, 23
gcs_load, 24, 27, 29, 30, 32
gcs_load_all, 13
gcs_load_all (gcs_save_all), 28
gcs_metadata_object, 4, 5, 11, 18, 23, 25, 34
gcs_parse_download, 12, 17, 26, 31
gcs_retry_upload, 26
gcs_save, 24, 27, 29, 30, 32
gcs_save_all, 13, 24, 27, 28, 30, 32
gcs_save_image, 24, 27, 29, 29, 32
gcs_setup, 30
gcs_signed_url, 12, 26, 31
gcs_source, 24, 27, 29, 30, 32
gcs_update_object_acl, 7, 12, 15, 19, 33
INDEX

- gcs_upload, 25–27, 34
- gcs_upload_set_limit, 35
- gcs_upload_set_limit(gcs_upload), 34
- gcs_version_bucket, 36
- googleCloudStorageR, 37

- save, 27
- save.image, 29
- source, 32

- toJSON, 35

- write.csv, 35

- zip, 28