

# Package ‘googleCloudStorageR’

May 8, 2026

**Type** Package

**Version** 0.7.0

**Title** Interface with Google Cloud Storage API

**Description** Interact with Google Cloud Storage <<https://cloud.google.com/storage/>> API in R. Part of the 'cloudyr' <<https://cloudyr.github.io/>> project.

**URL** <https://code.markedmondson.me/googleCloudStorageR/>

**BugReports** <https://github.com/cloudyr/googleCloudStorageR/issues>

**Depends** R (>= 3.2.0)

**Imports** assertthat (>= 0.2.0), cli, curl, googleAuthR (>= 1.4.0), httr (>= 1.2.1), jsonlite (>= 1.0), openssl, utils, yaml, zip (>= 2.0.3)

**Suggests** fs, googleComputeEngineR, knitr, readr, rmarkdown, sodium, testthat, usethis

**License** MIT + file LICENSE

**VignetteBuilder** knitr

**RoxygenNote** 7.1.2

**Config/testthat/edition** 3

**Config/testthat/parallel** false

**NeedsCompilation** no

**Author** Mark Edmondson [aut, cre] (ORCID: <<https://orcid.org/0000-0002-8434-3881>>)

**Maintainer** Mark Edmondson <r@sunholo.com>

**Repository** CRAN

**Date/Publication** 2021-12-16 09:50:02 UTC

## Contents

gcs_auth . . . . .	2
gcs_compose_objects . . . . .	4

gcs_copy_object . . . . .	5
gcs_create_bucket . . . . .	6
gcs_create_bucket_acl . . . . .	7
gcs_create_lifecycle . . . . .	7
gcs_create_pubsub . . . . .	8
gcs_delete_bucket . . . . .	10
gcs_delete_object . . . . .	11
gcs_delete_pubsub . . . . .	11
gcs_download_url . . . . .	12
gcs_first . . . . .	13
gcs_get_bucket . . . . .	14
gcs_get_bucket_acl . . . . .	15
gcs_get_global_bucket . . . . .	16
gcs_get_object . . . . .	17
gcs_get_object_acl . . . . .	19
gcs_get_service_email . . . . .	20
gcs_global_bucket . . . . .	20
gcs_list_buckets . . . . .	21
gcs_list_objects . . . . .	22
gcs_list_pubsub . . . . .	23
gcs_load . . . . .	24
gcs_metadata_object . . . . .	25
gcs_parse_download . . . . .	26
gcs_retry_upload . . . . .	26
gcs_save . . . . .	27
gcs_save_all . . . . .	28
gcs_save_image . . . . .	29
gcs_setup . . . . .	30
gcs_signed_url . . . . .	31
gcs_source . . . . .	32
gcs_update_object_acl . . . . .	32
gcs_upload . . . . .	34
gcs_version_bucket . . . . .	36
googleCloudStorageR . . . . .	37
<b>Index</b>	<b>38</b>

---

gcs_auth	<i>Authenticate with Google Cloud Storage API</i>
----------	---

---

### Description

Authenticate with Google Cloud Storage API

### Usage

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

## Arguments

json_file	Authentication json file you have downloaded from your Google Project
token	An existing authentication 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, making this function unnecessary.

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. You will still need the two JSON files - the client JSON and the authentication key JSON. [gcs\\_setup](#) can help set-up the latter, the client JSON you will need to download from your Google Cloud Project.

If using JSON files from another source, ensure it has either "https://www.googleapis.com/auth/devstorage.full\_control" or "https://www.googleapis.com/auth/cloud-platform" scopes.

## Examples

```
## Not run:
# on first run, generate a auth key via gcs_setup()

# the json file for the auth key you are using
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 .Renvirom 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-authorized 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`     *Compose up to 32 objects into one*

---

### Description

This merges objects stored on Cloud Storage into one object.

### Usage

```
gcs_compose_objects(objects, destination, bucket = gcs_get_global_bucket())
```

### Arguments

<code>objects</code>	A character vector of object names to combine
<code>destination</code>	Name of the new object.
<code>bucket</code>	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)
```

---

gcs_copy_object	<i>Copy an object</i>
-----------------	-----------------------

---

**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

```
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 <a href="#">gcs_create_lifecycle</a> 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 <a href="#">gcs_create_bucket</a>
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	<i>Create a pub/sub notification for a bucket</i>
-------------------	---

---

**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	<i>Delete a bucket</i>
-------------------	------------------------

---

### Description

Delete the bucket, and all its objects

### Usage

```
gcs_delete_bucket(  
    bucket,  
    ifMetagenerationMatch = NULL,  
    ifMetagenerationNotMatch = NULL,  
    force_delete = FALSE  
)  
  
gcs_delete_bucket_objects(bucket, include_versions = FALSE)
```

### Arguments

bucket	Name of the bucket, or a bucket object
ifMetagenerationMatch	Delete only if metageneration matches
ifMetagenerationNotMatch	Delete only if metageneration does not match
force_delete	If the bucket contains objects it will prevent deletion, including objects in a versioned bucket that previously existed. Setting this to TRUE will force deletion of those objects before deleting the bucket itself.
include_versions	Whether to include all historic versions of the objects to delete

### See Also

[gcs\\_delete\\_object](#)

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**

```
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**

To delete all objects in a bucket see [gcs\\_delete\\_bucket\\_objects](#)

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>

Other pubsub functions: [gcs\\_create\\_pubsub\(\)](#), [gcs\\_get\\_service\\_email\(\)](#), [gcs\\_list\\_pubsub\(\)](#)

---

gcs_download_url	<i>Get the download URL</i>
------------------	-----------------------------

---

**Description**

Create the download URL for objects in buckets

**Usage**

```
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**

```
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] `load_dir` - 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 `GCS_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 `load_dir` 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

[gcs\\_save\\_all](#) and [gcs\\_load\\_all](#) that these functions call

### Examples

```
## Not run:

.First <- function(){
  googleCloudStorageR::gcs_first()
}

.Last <- function(){
  googleCloudStorageR::gcs_last()
}

## End(Not run)
```

---

`gcs_get_bucket`

*Get bucket info*

---

### Description

Meta data about the bucket

### Usage

```
gcs_get_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**

```
## 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**

```
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 <a href="#">gcs_create_bucket</a>
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**

```
## 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**

```
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	<i>Get an object in a bucket directly</i>
----------------	---

---

**Description**

This retrieves an object directly.

**Usage**

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

**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 <a href="#">gcs_parse_download</a>
generation	The generation number for the noncurrent version, if you have object versioning enabled in your bucket e.g. "1560468815691234"

**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**

```
## 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.csv", parseFunction = f)

## download an RDS file using helper gcs_parse_rds()

gcs_get_object("obj.rds", parseFunction = gcs_parse_rds)

## to download from a folder in your bucket
my_folder <- "your_folder/"
objs <- gcs_list_objects(prefix = my_folder)

dir.create(my_folder)

# download all the objects to that folder
dls <- lapply(objs$name, function(x) gcs_get_object(x, saveToDisk = x))

## 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

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

### Arguments

<code>object_name</code>	Name of the object
<code>bucket</code>	Name of a bucket
<code>entity</code>	The entity holding the permission. Not needed for <code>entity_type</code> <code>allUsers</code> or <code>allAuthenticatedUsers</code>
<code>entity_type</code>	The type of entity
<code>generation</code>	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

```
## Not run:  
  
# 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: [gcs\\_create\\_pubsub\(\)](#), [gcs\\_delete\\_pubsub\(\)](#), [gcs\\_list\\_pubsub\(\)](#)

---

`gcs_global_bucket`    *Set global bucket name*

---

### Description

Set a bucket name used for this R session

### Usage

```
gcs_global_bucket(bucket)
```

### 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: [gcs\\_create\\_bucket\(\)](#), [gcs\\_create\\_lifecycle\(\)](#), [gcs\\_delete\\_bucket\(\)](#), [gcs\\_get\\_bucket\(\)](#), [gcs\\_get\\_global\\_bucket\(\)](#), [gcs\\_list\\_buckets\(\)](#)

---

gcs_list_buckets	<i>List buckets</i>
------------------	---------------------

---

**Description**

List the buckets your projectId has access to

**Usage**

```
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,
  versions = FALSE
)
```

**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.
versions	If TRUE, lists all versions of an object as distinct results in order of increasing generation number.

**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_list_pubsub	<i>List pub/sub notifications for a bucket</i>
-----------------	--

---

**Description**

List notification configurations for a bucket.

**Usage**

```
gcs_list_pubsub(bucket = gcs_get_global_bucket())
```

**Arguments**

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.

**See Also**

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

Other pubsub functions: [gcs\\_create\\_pubsub\(\)](#), [gcs\\_delete\\_pubsub\(\)](#), [gcs\\_get\\_service\\_email\(\)](#)

---

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**

```
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

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

### Arguments

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

```
gcs_parse_rds(object)
```

### Arguments

<code>object</code>	The object downloaded
<code>encoding</code>	Default to UTF-8

### Details

`gcs_parse_rds` will parse `.rds` files created via [saveRDS](#) without saving to disk

### See Also

[gcs\\_get\\_object](#)

Other download functions: [gcs\\_download\\_url\(\)](#), [gcs\\_signed\\_url\(\)](#)

---

`gcs_retry_upload`      *Retry a resumable 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 <code>gcs_upload_retry</code> .
upload_url	As created in a failed upload via <a href="#">gcs_upload</a>
file	The file location to upload
type	The file type, guessed if NULL
	Either supply a retry object, or the <code>upload_url</code> , <code>file</code> and <code>type</code> manually yourself. The function will first check to see how much has been uploaded already, then try to send up the remaining bytes.

**Value**

If successful, an object metadata object, if not an `gcs_upload_retry` object.

---

<code>gcs_save</code>	<i>Save .RData objects to the Google Cloud</i>
-----------------------	--

---

**Description**

Performs [save](#) then saves it to Google Cloud Storage.

**Usage**

```
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 <code>.RData</code> )
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.

`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**

```
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**

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

```
## Not run:  
  
gcs_save_all(  
  directory = "path-to-all-images",  
  bucket = "my-bucket",  
  predefinedAcl = "bucketLevel")  
  
## End(Not run)
```

---

<code>gcs_save_image</code>	<i>Save an R session to the Google Cloud</i>
-----------------------------	--

---

**Description**

Performs [save.image](#) then saves it to Google Cloud Storage.

**Usage**

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

**Arguments**

<code>file</code>	Where to save the file in GCS and locally
<code>bucket</code>	Bucket to store objects in
<code>saveLocation</code>	Which folder in the bucket to save file
<code>envir</code>	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.

**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**

```
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**

```
## Not run:
```

```
library(googleCloudStorageR)
gcs_setup()
```

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

---

gcs_signed_url	<i>Create a signed URL</i>
----------------	----------------------------

---

### 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

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

### Arguments

meta_obj	A meta object from <a href="#">gcs_get_object</a>
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

```
## 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 <a href="#">source</a>

### Value

TRUE if successful

### See Also

Other R session data functions: [gcs\\_load\(\)](#), [gcs\\_save\\_all\(\)](#), [gcs\\_save\\_image\(\)](#), [gcs\\_save\(\)](#)

---

gcs\_update\_object\_acl *Change access to an object in a bucket*

---

### 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

**See Also**

[objectAccessControls](#) on Google API reference

Other Access control functions: [gcs\\_create\\_bucket\\_acl\(\)](#), [gcs\\_get\\_bucket\\_acl\(\)](#), [gcs\\_get\\_object\\_acl\(\)](#)

---

gcs_upload	<i>Upload a file of arbitrary type</i>
------------	--

---

**Description**

Upload up to 5TB

**Usage**

```
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(upload_limit = 5000000L)
```

**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 <a href="#">gcs_metadata_object</a>
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

**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](https://www.googleapis.com/auth/devstorage.read_write) or <https://www.googleapis.com/auth>

## 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
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

```
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

```

## 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)

```

---

`googleCloudStorageR` *googleCloudStorageR*

---

**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, 19  
gcs\_update\_object\_acl, 32

## \* 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, 11  
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, 20  
gcs\_list\_pubsub, 23

content, 17, 26

gar\_gce\_auth, 14

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, 15–17, 21, 22

gcs\_create\_bucket\_acl, 7, 16, 19, 33

gcs\_create\_lifecycle, 6, 7, 10, 15, 17, 21, 22

gcs\_create\_pubsub, 8, 12, 20, 24

gcs\_delete\_all (gcs\_save\_all), 28

gcs\_delete\_bucket, 6, 8, 10, 15, 17, 21, 22

gcs\_delete\_bucket\_objects, 11

gcs\_delete\_bucket\_objects  
(gcs\_delete\_bucket), 10

gcs\_delete\_object, 4, 5, 10, 11, 18, 23, 25

gcs\_delete\_pubsub, 9, 11, 20, 24

gcs\_download\_url, 12, 17, 26, 31

gcs\_first, 13

gcs\_get\_bucket, 6, 8, 10, 14, 17, 21, 22

gcs\_get\_bucket\_acl, 7, 15, 19, 33

gcs\_get\_global\_bucket, 6, 8, 10, 15, 16, 21, 22

gcs\_get\_object, 4, 5, 11, 17, 23, 25–27, 31

gcs\_get\_object\_acl, 7, 16, 19, 33

gcs\_get\_service\_email, 9, 12, 20, 24

gcs\_global\_bucket, 6, 8, 10, 15–17, 20, 22

gcs\_last (gcs\_first), 13

gcs\_list\_buckets, 6, 8, 10, 15, 17, 21, 21

gcs\_list\_objects, 4, 5, 11, 18, 22, 25

gcs\_list\_pubsub, 9, 12, 20, 23

gcs\_load, 24, 27–30, 32

gcs\_load\_all, 14

gcs\_load\_all (gcs\_save\_all), 28

gcs\_metadata\_object, 4, 5, 11, 18, 23, 25, 34

gcs\_parse\_download, 13, 17, 26, 31

gcs\_parse\_rds (gcs\_parse\_download), 26

gcs\_retry\_upload, 26

gcs\_save, 24, 27, 29, 30, 32

gcs\_save\_all, 14, 24, 28, 28, 30, 32

gcs\_save\_image, 24, 27–29, 29, 32

gcs\_setup, [3](#), [30](#)  
gcs\_signed\_url, [13](#), [26](#), [31](#)  
gcs\_source, [24](#), [28–30](#), [32](#)  
gcs\_update\_object\_acl, [7](#), [12](#), [16](#), [19](#), [32](#)  
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](#)  
saveRDS, [26](#)  
source, [32](#)

toJSON, [35](#)

write.csv, [35](#)

zip, [28](#)