

Package ‘gdscloud’

September 16, 2026

Type Package

Title Cloud Storage Access for GDS Files

Version 0.99.4

Description Provides read-only access to GDS (Genomic Data Structure) files stored on cloud storage services including Amazon S3, Google Cloud Storage (GCS), and Azure Blob Storage, as well as any HTTP/HTTPS URL. It extends the 'gdsfmt' package so that cloud URLs (`http://`, `https://`, `s3://`, `gs://`, `az://`) can be opened transparently, without downloading the whole file first. Only the blocks that are actually read are fetched, using HTTP Range requests via libcurl together with an in-memory least-recently-used block cache, so that random access to a remote GDS file behaves like access to a local one. Credentials are resolved from the usual environment variables of each service, or set per session and per URL prefix, and they can be exported to the workers of a parallel cluster.

Depends R ($\geq 4.5.0$), gdsfmt ($\geq 1.49.7$)

Encoding UTF-8

LinkingTo gdsfmt

Suggests BiocParallel, BiocStyle, keyring, knitr, rmarkdown, testthat, SeqArray

VignetteBuilder knitr

License LGPL-3

SystemRequirements libcurl ($\geq 7.28.0$), OpenSSL

URL <https://github.com/zhengxwen/gdscloud>

BugReports <https://github.com/zhengxwen/gdscloud/issues>

biocViews Infrastructure, DataImport

git_url <https://git.bioconductor.org/packages/gdscloud>

git_branch devel

git_last_commit 0a4e7b0

git_last_commit_date 2026-08-31

Repository Bioconductor 3.24

Date/Publication 2026-09-15

Author Xiuwen Zheng [aut, cre] (ORCID:
[<https://orcid.org/0000-0002-1390-0708>](https://orcid.org/0000-0002-1390-0708))

Maintainer Xiuwen Zheng <zhengx@u.washington.edu>

Contents

gdscloud-package	2
gdsCloudCacheSize	3
gdsCloudConfigHTTP	4
gdsCloudConfigS3	5
gdsCloudExportCredentials	7
gdsCloudList	8
gdsCloudOpen	9
gdsCloudSchemes	10
Index	11

gdscloud-package	<i>Cloud Storage Access for GDS Files</i>
------------------	---

Description

gdscloud extends the **gdsfmt** package to provide transparent, read-only access to GDS (Genomic Data Structure) files stored on cloud storage services – Amazon S3 (`s3://`), Google Cloud Storage (`gs://`), and Azure Blob Storage (`az://`) – as well as any HTTP/HTTPS URL. It uses HTTP Range requests via `libcurl` together with an in-memory LRU block cache to minimize network overhead. Once the package is loaded, `openfn.gds` automatically recognizes cloud URLs and opens them transparently.

Details

Main functions:

- `gdsCloudOpen`: open a GDS file from a cloud URL.
- `gdsCloudSchemes`: list the supported URL schemes.
- `gdsCloudConfigS3`, `gdsCloudConfigGCS`, `gdsCloudConfigAzure`, `gdsCloudConfigHTTP`: configure credentials.
- `gdsCloudExportCredentials`: propagate credentials to parallel workers.
- `gdsCloudCacheSize`, `gdsCloudList`: control the block cache and inspect open cloud streams.

Author(s)

Xiuwen Zheng <zhengx@u.washington.edu>

See Also

`gdsCloudOpen`, `openfn.gds`

Examples

```
# list the supported cloud URL schemes
gdsCloudSchemes()
```

gdsCloudCacheSize	<i>Cache Control for Cloud GDS Access</i>
-------------------	---

Description

Manage the block cache used for cloud storage access.

Usage

```
gdsCloudCacheSize(size_mb=64)
gdsCloudCacheClear()
gdsCloudCacheInfo(verbose=TRUE)
```

Arguments

size_mb	default cache size in megabytes for new cloud streams
verbose	if TRUE, print cache statistics to the console

Details

Cloud GDS access uses an LRU block cache (1 MB blocks) to minimize HTTP requests. The default cache size is 64 MB per stream.

`gdsCloudCacheSize` sets the default cache size for subsequently opened streams.

`gdsCloudCacheClear` frees cached data.

`gdsCloudCacheInfo` prints cache statistics.

Value

`gdsCloudCacheSize`: the new cache size (invisible). `gdsCloudCacheClear`: none (invisible).
`gdsCloudCacheInfo`: a list with cache statistics (invisible).

See Also

[gdsCloudOpen](#)

Examples

```
# Increase cache to 128 MB
gdsCloudCacheSize(128)

# Show cache info
gdsCloudCacheInfo()
```

gdsCloudConfigHTTP	<i>Configure HTTP/HTTPS Credentials</i>
--------------------	---

Description

Set an optional Bearer token for authenticated HTTP/HTTPS access. Credentials set via this function take priority over environment variables. URL-specific credentials can also be registered so that different URLs use different tokens.

Usage

```
gdsCloudConfigHTTP(bearer_token=NULL, url=NULL)
```

Arguments

bearer_token	a Bearer token string for HTTP Authorization header (e.g. an OAuth2 access token or API token)
url	optional URL prefix (e.g. "https://private.example.com/") to associate the given credentials with. When NULL (the default) the credentials are stored as the global defaults. Passing url with bearer_token=NULL removes the entry for that prefix. The scheme must be http:// or https://.

Details

If not set explicitly, the Bearer token falls back to the GDSCLOUD_HTTP_TOKEN environment variable. When a URL is opened via [gdsCloudOpen](#), credentials are resolved in the following order (the first non-empty value wins):

1. the URL-specific entry whose registered prefix is the longest prefix of the opened URL;
2. the global value set by this function with url=NULL;
3. the GDSCLOUD_HTTP_TOKEN environment variable.

If no token is configured, the request is sent without an Authorization header (suitable for public URLs).

Value

None (invisible).

See Also

[gdsCloudOpen](#), [gdsCloudConfigS3](#)

Examples

```
# Set a global Bearer token
gdsCloudConfigHTTP(bearer_token = "my-secret-token")

## Not run:
# Set a URL-specific token
gdsCloudConfigHTTP(
  bearer_token = "token-for-private-server",
```

```

    url = "https://private.example.com/"
  )

  # Open a public HTTPS file (no token needed)
  gds <- gdsCloudOpen("https://example.com/data/file.gds")
  closefn.gds(gds)

  ## End(Not run)

```

gdsCloudConfigS3

*Configure Cloud Storage Credentials***Description**

Set authentication credentials for cloud storage access. Credentials set via these functions take priority over environment variables. URL-specific credentials can also be registered so that different URLs (e.g. different buckets) use different keys.

Usage

```

gdsCloudConfigS3(aws_access_key_id=NULL, aws_secret_access_key=NULL,
  region=NULL, session_token=NULL, url=NULL)

gdsCloudConfigGCS(access_token=NULL, url=NULL)

gdsCloudConfigAzure(account_name=NULL, account_key=NULL, sas_token=NULL,
  url=NULL)

```

Arguments

aws_access_key_id	AWS access key ID
aws_secret_access_key	AWS secret access key
region	AWS region (default: "us-east-1")
session_token	AWS session token for temporary credentials
access_token	Google Cloud OAuth2 access token
account_name	Azure storage account name
account_key	Azure storage account key
sas_token	Azure SAS (Shared Access Signature) token
url	optional URL prefix (e.g. "s3://my-bucket/") to associate the given credentials with. When NULL (the default) the credentials are stored as the global defaults. Passing url with all credential arguments NULL removes the entry for that prefix. The URL scheme must match the function (s3:// for gdsCloudConfigS3, gs:// for gdsCloudConfigGCS, az:// for gdsCloudConfigAzure)

Details

If not set explicitly, credentials fall back to environment variables:

- S3: AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_DEFAULT_REGION, AWS_SESSION_TOKEN
- GCS: GCS_ACCESS_TOKEN
- Azure: AZURE_STORAGE_ACCOUNT, AZURE_STORAGE_KEY, AZURE_STORAGE_SAS_TOKEN

When a URL is opened via [gdsCloudOpen](#), credentials are resolved in the following order (the first non-empty value wins for each field):

1. the URL-specific entry whose registered prefix is the longest prefix of the opened URL (within the same scheme);
2. the global values set by these functions with url=NULL;
3. the corresponding environment variable.

Registered URL prefixes are normalized by appending a trailing "/" if missing, so "s3://bucket" and "s3://bucket/" behave identically.

Value

None (invisible).

See Also

[gdsCloudOpen](#)

Examples

```
# AWS S3
gdsCloudConfigS3(
  aws_access_key_id = "your_key",
  aws_secret_access_key = "your_secret",
  region = "us-east-1"
)

# Google Cloud Storage
gdsCloudConfigGCS(access_token = "ya29.example_token")

# Azure Blob Storage
gdsCloudConfigAzure(
  account_name = "mystorageaccount",
  account_key = "base64encodedkey=="
)

## Not run:
# URL-specific credentials: different keys for different buckets
gdsCloudConfigS3(
  aws_access_key_id = "KEY_A",
  aws_secret_access_key = "SECRET_A",
  url = "s3://bucket-a/"
)
gdsCloudConfigS3(
  aws_access_key_id = "KEY_B",
  aws_secret_access_key = "SECRET_B",
  url = "s3://bucket-b/"
)
```

```
)  
# Remove a previously registered URL-specific entry  
gdsCloudConfigS3(url = "s3://bucket-a/")  
  
## End(Not run)
```

`gdsCloudExportCredentials`*Export Cloud Credentials to Child Processes*

Description

Copy cloud storage credentials configured in the current R session to worker processes of a parallel cluster, typically the one used by `SeqArray::seqParallel()`. This is required for PSOCK / SOCK / MPI / BiocParallel clusters, which do not inherit the parent R session's state.

Usage

```
gdsCloudExportCredentials(cl)
```

Arguments

`cl` a cluster specification following the convention of the `cl` argument to [seqParallel](#): NULL or FALSE (serial processing, no-op); TRUE or a numeric value (forking on Unix); a cluster object from the **parallel** package (e.g. created by [makeCluster](#)); or a `BiocParallelParam` object from the **BiocParallel** package.

Details

Credentials set via [gdsCloudConfigS3](#), [gdsCloudConfigGCS](#) and [gdsCloudConfigAzure](#) are stored in an internal environment of the **gdscloud** package in the parent R session, and are *not* automatically visible to workers spawned by socket-based clusters. Call `gdsCloudExportCredentials(cl)` once after creating the cluster and configuring credentials in the parent session, and before invoking [seqParallel](#).

On Unix, when `cl` is TRUE or a numeric value, `seqParallel()` uses forking and child processes inherit the parent's in-memory credentials automatically, so this function is a no-op in that case. On Windows, forking is not supported and an explicit cluster object should be created and passed to this function.

Environment variables (e.g. `AWS_ACCESS_KEY_ID`) on the worker machines are *not* modified.

Value

Invisibly returns TRUE if credentials were exported (or inherited via forking), and FALSE otherwise.

See Also

[gdsCloudConfigS3](#), [gdsCloudConfigGCS](#), [gdsCloudConfigAzure](#), [gdsCloudOpen](#)

Examples

```
# Serial processing (NULL): no-op, returns FALSE invisibly
gdsCloudExportCredentials(NULL)

library(parallel)

# configure credentials in the parent session
gdsCloudConfigS3(aws_access_key_id="AKID", aws_secret_access_key="secret")

# create a PSOCK cluster and export credentials to the workers
cl <- makeCluster(2L)
gdsCloudExportCredentials(cl)
stopCluster(cl)
```

gdsCloudList

*List Open Cloud Streams***Description**

List all currently open cloud GDS file streams with per-stream details including URL, file size, and cache statistics.

Usage

```
gdsCloudList()
```

Value

A data frame with one row per open cloud stream and columns:

url	the cloud URL (e.g., "s3://bucket/key")
file_size	total file size in bytes
cache_blocks	number of cached blocks currently held
cache_hits	cumulative cache hit count
cache_misses	cumulative cache miss count

If no streams are open, an empty data frame is returned.

See Also

[gdsCloudOpen](#), [gdsCloudCacheInfo](#)

Examples

```
# List currently open cloud streams (empty if none open)
gdsCloudList()

gds <- gdsCloudOpen("s3://gds-stat/download/hapmap/hapmap_r23a.gds")
gdsCloudList()
closefn.gds(gds)
```

gdsCloudOpen	<i>Open a GDS File from Cloud Storage</i>
--------------	---

Description

Opens a read-only GDS (Genomic Data Structure) file from cloud storage services including Amazon S3, Google Cloud Storage, and Azure Blob Storage, or from any HTTP/HTTPS URL.

Usage

```
gdsCloudOpen(url)
```

Arguments

url	a character string specifying the cloud URL, e.g., "s3://bucket/key", "gs://bucket/key", "az://container/blob", or "https://example.com/path/to/file.gds"
-----	---

Details

The function automatically detects the URL scheme and uses the appropriate cloud backend. Authentication credentials are read from environment variables or from values set via [gdsCloudConfigS3](#), [gdsCloudConfigHTTP](#), etc.

All cloud access is read-only. The file data is cached in memory using an LRU block cache (default 64 MB) for efficient random access.

When the `gdscloud` package is loaded, `openfn.gds()` from the `gdsfmt` package automatically detects cloud URLs and delegates to this function.

Value

An object of class `gds.class`, identical to the return value of [openfn.gds](#).

See Also

[gdsCloudConfigS3](#), [gdsCloudConfigHTTP](#), [gdsCloudCacheSize](#), [openfn.gds](#), [closefn.gds](#)

Examples

```
# Set AWS credentials
gdsCloudConfigS3(
  aws_access_key_id = "AKIAIOSFODNN7EXAMPLE",
  aws_secret_access_key = "wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY",
  region = "us-east-1"
)

## Not run:
# Open from S3
gds <- gdsCloudOpen("s3://my-bucket/data/example.gds")
read.gdsn(index.gdsn(gds, "genotype"))
closefn.gds(gds)

# Or transparently via openfn.gds (when gdscloud is loaded)
gds <- openfn.gds("s3://my-bucket/data/example.gds")
```

```

closefn.gds(gds)

# Open from an HTTP/HTTPS URL (public, no auth needed)
gds <- gdsCloudOpen("https://example.com/data/example.gds")
closefn.gds(gds)

# Authenticated HTTP endpoint
gdsCloudConfigHTTP(bearer_token = "my-secret-token")
gds <- gdsCloudOpen("https://private.example.com/data/example.gds")
closefn.gds(gds)

## End(Not run)

```

gdsCloudSchemes	<i>List Supported Cloud URL Schemes</i>
-----------------	---

Description

Returns the cloud URL schemes supported by gdscloud, with their corresponding storage service names.

Usage

```
gdsCloudSchemes()
```

Value

A named character vector where names are the URL scheme prefixes ("s3", "gs", "az", "http", "https") and values are the storage service descriptions.

See Also

[gdsCloudOpen](#)

Examples

```

gdsCloudSchemes()
##           s3                gs                az
## "Amazon S3" "Google Cloud Storage" "Azure Blob Storage"
##      http      https
##    "HTTP"    "HTTPS"

```

Index

- * **GDS**
 - gdsCloudCacheSize, [3](#)
 - gdsCloudConfigHTTP, [4](#)
 - gdsCloudConfigS3, [5](#)
 - gdsCloudExportCredentials, [7](#)
 - gdsCloudList, [8](#)
 - gdsCloudOpen, [9](#)
 - gdsCloudSchemes, [10](#)
- * **cloud**
 - gdsCloudCacheSize, [3](#)
 - gdsCloudConfigHTTP, [4](#)
 - gdsCloudConfigS3, [5](#)
 - gdsCloudExportCredentials, [7](#)
 - gdsCloudList, [8](#)
 - gdsCloudOpen, [9](#)
 - gdsCloudSchemes, [10](#)
- * **package**
 - gdscld-cloud, [2](#)
- * **parallel**
 - gdsCloudExportCredentials, [7](#)
- closefn.gds, [9](#)
- gdscld (gdscld-cloud), [2](#)
- gdscld-cloud, [2](#)
- gdsCloudCacheClear (gdsCloudCacheSize),
[3](#)
- gdsCloudCacheInfo, [8](#)
- gdsCloudCacheInfo (gdsCloudCacheSize), [3](#)
- gdsCloudCacheSize, [2](#), [3](#), [9](#)
- gdsCloudConfigAzure, [2](#), [7](#)
- gdsCloudConfigAzure (gdsCloudConfigS3),
[5](#)
- gdsCloudConfigGCS, [2](#), [7](#)
- gdsCloudConfigGCS (gdsCloudConfigS3), [5](#)
- gdsCloudConfigHTTP, [2](#), [4](#), [9](#)
- gdsCloudConfigS3, [2](#), [4](#), [5](#), [7](#), [9](#)
- gdsCloudExportCredentials, [2](#), [7](#)
- gdsCloudList, [2](#), [8](#)
- gdsCloudOpen, [2-4](#), [6-8](#), [9](#), [10](#)
- gdsCloudSchemes, [2](#), [10](#)
- makeCluster, [7](#)
- openfn.gds, [2](#), [9](#)
- seqParallel, [7](#)