Package: gdscloud (via r-universe)

November 25, 2026

Type Package

Title Cloud Storage Access for GDS Files

Version 1.0.0

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 (>= 4.5.0), gdsfmt (>= 1.49.7)

Encoding UTF-8

LinkingTo gdsfmt

Suggests BiocParallel, BiocStyle, keyring, knitr, rmarkdown, testthat, SeqArray, httpuv, callr, openssl, withr

VignetteBuilder knitr

License LGPL-3

SystemRequirements libcurl (>= 7.28.0; >= 7.32.0 recommended), OpenSSL (on Linux and other Unix-like systems; not needed on macOS or Windows)

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

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

biocViews Infrastructure, DataImport

Config/pak/sysreqs libssl-dev

Repository Bioconductor 3.24

Date/Publication 2026-11-23 22:56:56 UTC

RemoteUrl https://github.com/bioc/gdscloud

RemoteRef RELEASE_3_24

RemoteSha 0c11da1c49c677dfa669dddbdb277007480ddc89

Contents

gdscloud-package 2
gdsCloudCacheSize 3
gdsCloudConfigHTTP 4
gdsCloudConfigS3 5
gdsCloudExportCredentials 9
gdsCloudList 11
gdsCloudOpen 12
gdsCloudOptions 13
gdsCloudSchemes 15

Index **16**

`gdscloud-package` *Cloud Storage Access for GDS Files*

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.
- `gdsCloudOptions`: timeouts and the default cache size.
- `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 *Cache Control for Cloud GDS Access*

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 (the same setting as the cache_size option of gdsCloudOptions).

gdsCloudCacheClear frees cached data.

gdsCloudCacheInfo prints cache statistics: the number of open cloud streams, cache hits and misses summed over the open streams, and the number of requests that were retried after a transient failure (see gdsCloudOptions).

Value

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

See Also

gdsCloudOpen, gdsCloudOptions

Examples

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

# Show cache info
gdsCloudCacheInfo()
```

`gdsCloudConfigHTTP` *Configure HTTP/HTTPS Credentials*

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), or a function with no arguments returning it; a function is called each time a file is opened, so expiring tokens can be refreshed (see `gdsCloudConfigS3` for details)
`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:
# A token provider function, called whenever a file is opened
gdsCloudConfigHTTP(bearer_token = function() my_oauth_client$get_token())

# 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, credentials=NULL, endpoint=NULL,
    path_style=NULL, url=NULL)

gdsCloudConfigGCS(access_token=NULL, url=NULL)

gdsCloudConfigAzure(account_name=NULL, account_key=NULL, sas_token=NULL,
    access_token=NULL, endpoint_suffix=NULL, endpoint=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

`credentials`
: a function with no arguments returning a named list with `aws_access_key_id`, `aws_secret_access_key` and, for temporary credentials, `session_token` (optionally also `region`), or `NULL` when no credentials are available; called each time a file is opened. See 'Token provider functions' below

`endpoint`
: for `gdsCloudConfigS3`: base URL of an S3-compatible service, e.g. `"https://minio.example.org:90` or `"https://<account>.r2.cloudflarestorage.com"`; `""` or unset selects Amazon S3 (`https://<bucket>.s3.<region>.amazonaws.com`). For `gdsCloudConfigAzure`: full base URL of the blob service, overriding the account host, e.g. `"http://127.0.0.1:10000/devstor` for the Azurite emulator; `az://container/blob` is then served from `<endpoint>/container/blob`. In both cases the scheme defaults to `https://` when omitted and a path prefix is allowed.

`path_style`
: `TRUE` to address objects as `https://<host>/<bucket>/<key>` (path style), `FALSE` for `https://<bucket>.<host>/<key>` (virtual-hosted style), `NA` (the default) for path style with a custom `endpoint` and virtual-hosted style with Amazon S3

`access_token`
: for `gdsCloudConfigGCS`: a Google Cloud OAuth2 access token. For `gdsCloudConfigAzure`: an Azure OAuth2 access token for the storage resource (Microsoft Entra ID, e.g. from `az account get-access-token --resource https://storage.azure.com/` or a managed identity); used when no SAS token is set. Either may be a function returning the token, see 'Token provider functions' below

`account_name`
: Azure storage account name

`account_key`
: Azure storage account key

`sas_token`
: Azure SAS (Shared Access Signature) token

`endpoint_suffix`
: DNS suffix of the blob service, default `"blob.core.windows.net"`; e.g. `"blob.core.chinacloudapi.c` (Azure China) or `"blob.core.usgovcloudapi.net"` (Azure Government)

`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`; `AWS_ENDPOINT_URL_S3` or `AWS_ENDPOINT_URL` for `endpoint`, `GDSCLOUD_S3_PATH_STYLE` (`true`/`false`) for `path_style`
- GCS: `GCS_ACCESS_TOKEN`
- Azure: `AZURE_STORAGE_ACCOUNT`, `AZURE_STORAGE_KEY`, `AZURE_STORAGE_SAS_TOKEN`, `AZURE_STORAGE_ACCESS_TOKE` `AZURE_STORAGE_ENDPOINT_SUFFIX`, `AZURE_STORAGE_SERVICE_ENDPOINT`; and, as a last resort, the fields of `AZURE_STORAGE_CONNECTION_STRING` (`AccountName`, `AccountKey`, `SharedAccessSignature`, `EndpointSuffix`, `BlobEndpoint`, `UseDevelopmentStorage=true`)

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.

Token provider functions: The token arguments, `access_token` of `gdsCloudConfigGCS` and `gdsCloudConfigAzure` and `bearer_token` of gdsCloudConfigHTTP, may also be a function with no arguments that returns the token as a single character string. The function is called each time a file is opened, so tokens that expire (Google OAuth2 and Microsoft Entra ID access tokens last about an hour) are refreshed without reconfiguring. Returning `NULL` or `""` means no token is available and the next fallback (global setting, environment variable) is tried. A function is only called when the file is opened; requests made later on the same open file keep using the token obtained then.

AWS temporary credentials (STS, IAM Identity Center / SSO, EC2 or EKS instance roles) are a set of three values that expire together, so `gdsCloudConfigS3` takes one `credentials` function returning the whole set as a named list instead of per-field functions. When it returns an access key, the set is used as is (no mixing with static values); when it returns `NULL` or no access key, the static settings and environment variables are used. Static AWS access keys, the Azure account key and SAS tokens are long-lived and are given as strings.

This is the intended way to plug in the login machinery of dedicated packages and tools, e.g. **gargle** or **googleAuthR** for Google, **AzureAuth** for Azure, and the AWS CLI (`aws configure export-credentials`) or **aws.signature** for AWS; see the examples.

Azure authentication: When several Azure credentials are configured, a SAS token is used first, then an OAuth2 `access_token`, then the Shared Key `account_key`; with none of them the request is anonymous (public containers). Requests to a sovereign cloud only need `endpoint_suffix`; a self-hosted or emulated service (Azurite) needs the full `endpoint`.

S3-compatible services: Any service that implements the S3 API with AWS Signature Version 4 can be used through `endpoint`: MinIO, Ceph RGW, Cloudflare R2, Wasabi, Backblaze B2, DigitalOcean Spaces, and the S3 interfaces of Alibaba OSS, Tencent COS, Oracle OCI and IBM COS, among others. `s3://bucket/key` then refers to the bucket on that service. Most of these expect path-style addressing, which is the default for a custom endpoint; `region` is still part of the signature and should be set to whatever the service documents (`"us-east-1"` or `"auto"` is accepted by most). A bucket name containing dots on Amazon S3 needs `path_style=TRUE`, because virtual-hosted HTTPS would fail the certificate check.

Value

None (invisible).

See Also

gdsCloudOpen, gdsCloudConfigHTTP, gdsCloudExportCredentials

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:
# Token provider functions: called every time a file is opened
# Google Cloud Storage via gargle (user or service-account login)
gdsCloudConfigGCS(access_token = function() {
    scope <- "https://www.googleapis.com/auth/devstorage.read_only"
    gargle::token_fetch(scopes = scope)$credentials$access_token
})

# ... or via the gcloud CLI
gdsCloudConfigGCS(access_token = function()
    system2("gcloud", c("auth", "print-access-token"), stdout = TRUE))

# Azure via AzureAuth (Entra ID)
gdsCloudConfigAzure(account_name = "mystorageaccount", access_token = function()
    AzureAuth::get_azure_token("https://storage.azure.com/",
        tenant = "mytenant", app = "myapp")$credentials$access_token)

# AWS temporary credentials from the AWS CLI's credential chain
# (profiles, SSO, assume-role, instance roles): the whole set at once
gdsCloudConfigS3(credentials = function() {
    out <- system2("aws", c("configure", "export-credentials",
        "--format", "process"), stdout = TRUE)
    j <- jsonlite::fromJSON(paste(out, collapse = ""))
    list(aws_access_key_id = j$AccessKeyId,
        aws_secret_access_key = j$SecretAccessKey,
        session_token = j$SessionToken)
})

# ... or via aws.signature (field names are accepted as returned)
gdsCloudConfigS3(credentials = function() {
    cr <- aws.signature::locate_credentials()
    list(aws_access_key_id = cr$key, aws_secret_access_key = cr$secret,
        session_token = cr$session_token, region = cr$region)
})
```

```
# 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/")

# An S3-compatible service (MinIO on a local network)
gdsCloudConfigS3(
    aws_access_key_id = "minioadmin",
    aws_secret_access_key = "minioadmin",
    endpoint = "http://minio.lab.internal:9000"
)
gds <- gdsCloudOpen("s3://genomics/hapmap.gds")
closefn.gds(gds)

# Azure with an Entra ID token, in a sovereign cloud
gdsCloudConfigAzure(
    account_name = "mystorageaccount",
    access_token = "eyJ0eXAiOiJKV1QiLCJhbGci...",
    endpoint_suffix = "blob.core.chinacloudapi.cn"
)

# Azurite emulator (the well-known development account and key)
gdsCloudConfigAzure(
    account_name = "devstoreaccount1",
    account_key = paste0("Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6",
        "IFsuFq2UVErCz4I6tq/K1SZFPTOtr/KBHBeksoGMGw=="),
    endpoint = "http://127.0.0.1:10000/devstoreaccount1"
)

# Cloudflare R2, scoped to one bucket
gdsCloudConfigS3(
    aws_access_key_id = "R2_KEY",
    aws_secret_access_key = "R2_SECRET",
    region = "auto",
    endpoint = "https://<ACCOUNT_ID>.r2.cloudflarestorage.com",
    url = "s3://my-r2-bucket/"
)

## 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`.

Tokens given as provider functions and the AWS `credentials` function (see `gdsCloudConfigS3`) are serialized to the workers together with their enclosing environments, and are called on the worker when it opens a file; the packages and login state they rely on must therefore be available there as well.

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)

# forget the example credentials again
gdsCloudConfigS3(aws_access_key_id="", aws_secret_access_key="")
```

---

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()


# a public bucket, accessed anonymously (no credentials configured)
gdsCloudConfigS3(aws_access_key_id="", aws_secret_access_key="")
gds <- gdsCloudOpen("s3://gds-stat/download/hapmap/hapmap_r23a.gds")
gdsCloudList()
closefn.gds(gds)
```

`gdsCloudOpen` *Open a GDS File from Cloud Storage*

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.

Every request has a connect timeout (default 30 seconds) and a low-speed timeout: a transfer that moves less than one byte per second for the timeout period (default 60 seconds) is aborted with an error. Both can be changed with `gdsCloudOptions`. A transfer can be interrupted with Ctrl-C (or Esc in RStudio), which raises an error.

Value

An object of class `gds.class`, identical to the return value of `openfn.gds`.

See Also

gdsCloudConfigS3, gdsCloudConfigHTTP, gdsCloudOptions, 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)
```

---

gdsCloudOptions *Package Options for Cloud Access*

Description

Get or set the options that control cloud transfers: connection and low-speed timeouts, the number of retries, and the default block-cache size.

Usage

```
gdsCloudOptions(connect_timeout=NULL, timeout=NULL, max_retries=NULL,
    cache_size=NULL)
```

Arguments

connect_timeout	maximum time in seconds to establish a connection (default 30)
timeout	low-speed timeout in seconds: a transfer that moves less than one byte per second for this long is aborted with an error (default 60)
max_retries	how many times a request is retried after a transient failure (default 3; 0 disables retrying). Transient failures are connection resets, timeouts and the HTTP statuses 408, 429, 500, 502, 503 and 504. Retries wait 0.5, 1, 2, 4 and then 8 seconds between attempts.
cache_size	default block-cache size in megabytes for newly opened cloud streams (default 64); equivalent to gdsCloudCacheSize

Details

Called without arguments, the function returns the current settings. Arguments that are NULL are left unchanged. Timeouts are read each time a file is opened and then apply to every request of every open stream.

The initial values can be set persistently through R options or environment variables, e.g. in .Rprofile or .Renviron:

option	environment variable	default
gdscloud.connect_timeout	GDSCLOUD_CONNECT_TIMEOUT	30
gdscloud.timeout	GDSCLOUD_TIMEOUT	60
gdscloud.max_retries	GDSCLOUD_MAX_RETRIES	3
gdscloud.cache_size_mb	GDSCLOUD_CACHE_SIZE_MB	64

They are consulted once, when the package is loaded.

gdsCloudExportCredentials copies the current options to the workers of a parallel cluster together with the credentials.

Value

A list with the current settings (connect_timeout, timeout, max_retries, cache_size); invisibly when any argument was supplied.

See Also

gdsCloudOpen, gdsCloudCacheSize, gdsCloudExportCredentials

Examples

```
# show the current settings
gdsCloudOptions()

# a slow or distant server: allow more time before giving up
gdsCloudOptions(connect_timeout=60, timeout=300)

# fail fast instead of retrying
```

```
gdsCloudOptions(max_retries=0)

# restore the defaults
gdsCloudOptions(connect_timeout=30, timeout=60, max_retries=3, cache_size=64)
```

`gdsCloudSchemes` *List Supported Cloud URL Schemes*

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, 10
  gdsCloudList, 11
  gdsCloudOpen, 12
  gdsCloudOptions, 13
  gdsCloudSchemes, 15
∗ **cloud**
  gdsCloudCacheSize, 3
  gdsCloudConfigHTTP, 4
  gdsCloudConfigS3, 5
  gdsCloudExportCredentials, 10
  gdsCloudList, 11
  gdsCloudOpen, 12
  gdsCloudOptions, 13
  gdsCloudSchemes, 15
∗ **package**
  gdscloud-package, 2
∗ **parallel**
  gdsCloudExportCredentials, 10

closefn.gds, *13*

gdscloud *(*gdscloud-package*)*, 2
gdscloud-package, 2
gdsCloudCacheClear *(*gdsCloudCacheSize*)*, 3
gdsCloudCacheInfo, *11*
gdsCloudCacheInfo *(*gdsCloudCacheSize*)*, 3
gdsCloudCacheSize, *2*, 3, *13, 14*
gdsCloudConfigAzure, *2*, *10*
gdsCloudConfigAzure *(*gdsCloudConfigS3*)*, 5
gdsCloudConfigGCS, *2*, *10*
gdsCloudConfigGCS *(*gdsCloudConfigS3*)*, 5
gdsCloudConfigHTTP, *2*, 4, *7*, *12, 13*
gdsCloudConfigS3, *2*, *4*, 5, *5*, *10*, *12, 13*
gdsCloudExportCredentials, *2*, *7*, 9, *14*
gdsCloudList, *2*, 11
gdsCloudOpen, *2–5*, *7*, *10, 11*, 12, *14, 15*
gdsCloudOptions, *2, 3*, *12*, 13, *13*
gdsCloudSchemes, *2*, 15

makeCluster, *10*

openfn.gds, *2, 3*, *12, 13*

seqParallel, *10*