Package 'gypsum'

November 1, 2025

Version 1.6.0

Date 2024-06-20

Title Interface to the gypsum REST API

Description Client for the gypsum REST API (https://gypsum.artifactdb.com), a cloud-based file store in the ArtifactDB ecosystem.
This package provides functions for uploads, downloads, and various adminstrative and management tasks.
Check out the documentation at https://github.com/ArtifactDB/gypsum-worker for more details.

License MIT + file LICENSE

Imports utils, httr2, jsonlite, parallel, filelock, rappdirs

Suggests knitr, rmarkdown, testthat, BiocStyle, digest, jsonvalidate, DBI, RSQLite, S4Vectors, methods

RoxygenNote 7.3.1

VignetteBuilder knitr

URL https://github.com/ArtifactDB/gypsum-R

BugReports https://github.com/ArtifactDB/gypsum-R/issues

biocViews DataImport

git_url https://git.bioconductor.org/packages/gypsum

git_branch RELEASE_3_22

git_last_commit ff6acdc

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2025-10-31

Author Aaron Lun [aut, cre]

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

Contents

`abortUpload` *Abort an upload*

Description

Abort an upload session, usually after an irrecoverable error.

Usage

```
abortUpload(init, url = restUrl())
```

Arguments

init	List containing abort_url and session_token. This is typically the return value from startUpload.
url	String containing the URL of the gypsum REST API.

Value

NULL is invisibly returned on successful abort.

Author(s)

Aaron Lun

See Also

startUpload, to create init.

Examples

```
tmp <- tempfile()
dir.create(tmp)
write(file=file.path(tmp, "blah.txt"), LETTERS)
dir.create(file.path(tmp, "foo"))
write(file=file.path(tmp, "foo", "bar.txt"), 1:10)

if (interactive()) {
    init <- startUpload(
        project="test-R",
        asset="upload-abort-check",
        version="v1",
        files=list.files(tmp, recursive=TRUE),
        probation=TRUE,
        directory=tmp
    )

    # Aborting the upload.
    abortUpload(init)
}
```

---

accessToken *Get and set GitHub access tokens*

Description

Get and set GitHub access tokens for authentication to the gypsum API's endpoints.

Usage

```
accessToken(full = FALSE, request = TRUE, cache = cacheDirectory())

setAccessToken(
  token,
  app.url = restUrl(),
  app.key = NULL,
  app.secret = NULL,
  github.url = "https://api.github.com",
  user.agent = NULL,
  cache = cacheDirectory()
)
```

Arguments

full	Logical scalar indicating whether to return the full token details.
request	Logical scalar indicating whether to request a new token if no cached token is available or if the current token has expired.
cache	String containing a path to the cache directory, to store the token across R sessions. If NULL, the token is not cached to (or read from) disk, which improves security on shared filesystems.
token	String containing a GitHub personal access token. This should have the "read:org" and "read:user" scopes. If missing, the user will be prompted to use GitHub's Oauth web application flow to acquire a token. If NULL, any existing tokens are cleared from cache.
app.url	String containing a URL of the gypsum REST API. This is used to obtain app.key and app.secret if either are NULL.
app.key	String containing the key for a GitHub Oauth app.
app.secret	String containing the secret for a GitHub Oauth app.
github.url	String containing the URL for the GitHub API. This is used to acquire more information about the token.
user.agent	String specifying the user agent for queries to various endpoints.

Value

setAccessToken sets the access token and invisibly returns a list containing:

- token, a string containing the token.
- name, the name of the GitHub user authenticated by the token.
- expires, the Unix time at which the token expires.

If full=TRUE, accessToken returns the same list, typically retrieved from one of the caches. If no token was cached or the cached token has expired, it will call setAccessToken with default arguments to obtain one if request=TRUE; otherwise if request=FALSE, NULL is returned.

If full=FALSE, accessToken will return a string containing a token (or NULL, if no token is available and request=FALSE).

Author(s)

Aaron Lun

Examples

```
if (interactive()) {
    accessToken()
}
```

`approveProbation` *Approve a probational upload*

Description

Pretty much as it says: approve a probational upload of a version of a project's asset. This removes the `on_probation` tag from the uploaded version.

Usage

```
approveProbation(
  project,
  asset,
  version,
  url = restUrl(),
  token = accessToken()
)
```

Arguments

`project`	String containing the project name.
`asset`	String containing the asset name.
`version`	String containing the version name.
`url`	String containing the URL of the gypsum REST API.
`token`	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to an owner of `project`.

Value

`NULL` is invisibly returned upon successful approval.

Author(s)

Aaron Lun

See Also

`rejectProbation`, to reject the probational upload.

`startUpload`, to specify probational uploads.

Examples

```
if (interactive()) {
    # Mocking up a versioned asset.
    init <- startUpload(
        project="test-R",
        asset="probation-approve",
        version="v1",
        files=character(0),
        probation=TRUE
    )
    completeUpload(init)

    # Approving the probation:
    approveProbation("test-R", "probation-approve", "v1")

    # Just cleaning up after we're done.
    removeProjectAsset("test-R", "probation-approve")
}
```

`cacheDirectory` *Cache directory*

Description

Specify the cache directory in the local filesystem for gypsum-related data.

Usage

```
cacheDirectory(dir)
```

Arguments

`dir`	String containing the path to a cache directory.

Details

If the `GYPSUM_CACHE_DIR` environment variable is set before the first call to `cacheDirectory`, it is used as the initial location of the cache directory. Otherwise, the initial location is based on `R_user_dir`.

Value

If `dir` is missing, the current setting of the cache directory is returned.

If `dir` is provided, it is used to replace the current setting of the cache directory, and the *previous* setting is invisibly returned.

Author(s)

Aaron Lun

Examples

```
cacheDirectory()
old <- cacheDirectory(tempfile())
cacheDirectory()
cacheDirectory(old) # setting it back.
```

`cloneVersion` *Clone a version's directory structure*

Description

Clone the directory structure for a versioned asset into a separate location. This is typically used to prepare a new version for a lightweight upload.

Usage

```
cloneVersion(
  project,
  asset,
  version,
  destination,
  download = TRUE,
  cache = cacheDirectory(),
  url = restUrl(),
  config = NULL,
  ...
)
```

Arguments

`project`	String containing the project name.
`asset`	String containing the asset name.
`version`	String containing the version name.
`destination`	String containing a path to a destination directory at which to create the clone.
`download`	Logical scalar indicating whether the version's files should be downloaded first. This can be set to `FALSE` to create a clone without actually downloading any of the version's files.
`cache`	String containing the path to the cache directory.
`url`	String containing the URL of the gypsum REST API.
`config`	Deprecated and ignored.
`...`	Further arguments to pass to `saveVersion`. Only used if `download=TRUE`.

Details

Cloning of a versioned asset involves creating a directory at `destination` that has the same contents as the corresponding project-asset-version directory. All files in the specified version are represented as symlinks from `destination` to the corresponding file in the `cache`. The idea is that, when `destination` is used in `prepareDirectoryUpload`, the symlinks are converted into upload links, i.e., `links=` in `startUpload`. This allows users to create new versions very cheaply as duplicate files are not uploaded to/stored in the backend.

Users can more-or-less do whatever they want inside the cloned `destination`, but they should treat the symlink targets as read-only. That is, they should not modify the contents of the linked-to file, as these refer to assumed-immutable files in the `cache`. If a file in `destination` needs to be modified, the symlink should be deleted and replaced with an actual file; this avoids mutating the `cache` and it ensures that `prepareDirectoryUpload` recognizes that a new file actually needs to be uploaded.

Advanced users can set `download=FALSE`, in which case symlinks are created even if their targets are not present in `cache`. In such cases, `destination` should be treated as write-only due to the potential presence of dangling symlinks. This mode is useful for uploading a new version of an asset without downloading the files from the existing version, assuming that the modifications associated with the former can be achieved without reading any of the latter.

On Windows, the user may not have permissions to create symbolic links, so the function will transparently fall back to creating hard links or copies instead. This precludes any optimization by `prepareDirectoryUpload` as the hard links/copies cannot be converted into upload links. It also assumes that `download=TRUE` as dangling links/copies cannot be created.

Value

The directory structure of the specified version is cloned to `destination`, and a `NULL` is invisibly returned.

Author(s)

Aaron Lun

See Also

`prepareDirectoryUpload`, to prepare an upload based on the directory contents.

Examples

```
tmp <- tempfile()
out <- cloneVersion("test-R", "basic", "v1", destination=tmp)
list.files(tmp, recursive=TRUE)
Sys.readlink(file.path(tmp, "foo", "bar.txt"))

# Files should be replaced rather than modified via the symlink:
existing <- file.path(tmp, "foo", "bar.txt")
unlink(existing) # Deleting the symlink...
write(file=existing, "YAY") # ... and writing a replacement file.

# Symlinks are converted to upload links:
prepareDirectoryUpload(tmp)
```

completeUpload *Complete an upload*

Description

Complete an upload session after all files have been uploaded.

Usage

```
completeUpload(init, url = restUrl())
```

Arguments

init	List containing complete_url and session_token. This is typically the return value from startUpload.
url	String containing the URL of the gypsum REST API.

Value

NULL is invisibly returned on successful completion.

Author(s)

Aaron Lun

See Also

startUpload, to create init.

Examples

```
tmp <- tempfile()
dir.create(tmp)
write(file=file.path(tmp, "blah.txt"), LETTERS)
dir.create(file.path(tmp, "foo"))
write(file=file.path(tmp, "foo", "bar.txt"), 1:10)

if (interactive()) {
    init <- startUpload(
        project="test-R",
        asset="upload-complete-check",
        version="v1",
        files=list.files(tmp, recursive=TRUE),
        probation=TRUE,
        directory=tmp
    )
    uploadFiles(init, directory=tmp)

    # Finishing the upload.
    completeUpload(init)
}
```

createProject *Create a new project*

Description

Create a new project with the associated permissions.

Usage

```
createProject(
  project,
  owners,
  uploaders = list(),
  baseline = NULL,
  growth = NULL,
  year = NULL,
  url = restUrl(),
  token = accessToken()
)
```

Arguments

project	String containing the project name.
owners	Character vector containing the GitHub users or organizations that are owners of this project.
uploaders	List specifying the authorized uploaders for this project. See the uploaders field in the fetchPermissions return value for the expected format.
baseline	Numeric scalar specifying the baseline quota in bytes. If NULL, the backend's default is used.
growth	Numeric scalar specifying the quota's annual growth rate in bytes. If NULL, the backend's default is used.
year	Integer scalar specifying the year of the project creation. If NULL, the backend's default is used - this should be the current year.
url	String containing the URL of the gypsum REST API.
token	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to a gypsum administrator account.

Value

NULL is invisibly returned if the project was successfully created.

Author(s)

Aaron Lun

See Also

removeProject, to remove a project.

Examples

```
if (interactive()) {
    createProject(
        "test-R-create",
        owners="LTLA",
        uploaders=list(list(id="ArtifactDB-bot"))
    )
}
```

`fetchLatest` *Fetch the latest version*

Description

Fetch the latest version of a project's asset.

Usage

```
fetchLatest(project, asset, url = restUrl(), config = NULL)
```

Arguments

`project`	String containing the project name.
`asset`	String containing the asset name.
`url`	String containing the URL of the gypsum REST API.
`config`	Deprecated and ignored.

Value

String containing the latest version of the project.

Author(s)

Aaron Lun

See Also

`refreshLatest`, to refresh the latest version.

Examples

```
fetchLatest("test-R", "basic")
```

`fetchManifest` *Fetch version manifest*

Description

Fetch the manifest for a version of an asset of a project.

Usage

```
fetchManifest(
  project,
  asset,
  version,
  cache = cacheDirectory(),
  overwrite = FALSE,
  url = restUrl(),
  config = NULL
)
```

Arguments

`project`	String containing the project name.
`asset`	String containing the asset name.
`version`	String containing the version name.
`cache`	String containing the cache directory. If `NULL`, no caching is performed.
`overwrite`	Logical scalar indicating whether to overwrite an existing file in `cache`, if one is present.
`url`	String containing the URL of the gypsum REST API.
`config`	Deprecated and ignored.

Value

List containing the manifest for this version. Each element is named after the relative path of a file in this version. The value of each element is another list with the following fields:

- `size`, an integer specifying the size of the file in bytes.
- `md5sum`, a string containing the hex-encoded MD5 checksum of the file.
- `link` (optional): a list specifying the link destination for a file. This contains the strings `project`, `asset`, `version` and `path`. If the link destination is itself a link, an `ancestor` list will be present that specifies the final location of the file after resolving all intermediate links.

Author(s)

Aaron Lun

Examples

```
fetchManifest("test-R", "basic", "v1")
```

`fetchMetadataDatabase` *Fetch a metadata database*

Description

Fetch a SQLite database containing metadata from the gypsum backend (see https://github.com/ArtifactDB/bioconductor-metadata-index). Each database is generated by aggregating metadata across multiple assets and/or projects, and can be used to perform searches for interesting objects.

Usage

```
fetchMetadataDatabase(
  name = "bioconductor.sqlite3",
  cache = cacheDirectory(),
  overwrite = FALSE
)
```

Arguments

`name`	String containing the name of the database. This can be the name of any SQLite file in https://github.com/ArtifactDB/bioconductor-metadata-index/releases/tag/latest.
`cache`	String containing the cache directory. If `NULL`, no caching is performed.
`overwrite`	Logical scalar indicating whether to overwrite an existing file in `cache`, if one is present.

Details

This function will automatically check for updates to the SQLite files and will download new versions accordingly. New checks are performed when one hour or more has elapsed since the last check. If the check fails, a warning is raised and the function returns the currently cached file.

Value

String containing a path to the downloaded database.

Author(s)

Aaron Lun

See Also

`fetchMetadataSchema`, to get the JSON schema used to define the database tables.

Examples

```
fetchMetadataDatabase()
```

`fetchMetadataSchema` *Fetch a metadata schema*

Description

Fetch a JSON schema file for metadata to be inserted into a SQLite database (see https://github.com/ArtifactDB/bioconductor-metadata-index). Each SQLite database is created from metadata files uploaded to the gypsum backend, so clients uploading objects to be incorporated into the database should validate their metadata against the corresponding JSON schema.

Usage

```
fetchMetadataSchema(
  name = "bioconductor/v1.json",
  cache = cacheDirectory(),
  overwrite = FALSE
)
```

Arguments

`name`	String containing the name of the schema. This can be the name of any JSON schema file published at https://github.com/ArtifactDB/bioconductor-metadata-index.
`cache`	String containing the cache directory. If `NULL`, no caching is performed.
`overwrite`	Logical scalar indicating whether to overwrite an existing file in `cache`, if one is present.

Value

String containing a path to the downloaded schema.

Author(s)

Aaron Lun

See Also

`validateMetadata`, to validate metadata against a chosen schema.

`fetchMetadataDatabase`, to obtain the SQLite database of metadata.

Examples

```
fetchMetadataSchema()
```

`fetchPermissions` *Fetch project permissions*

Description

Fetch the permissions for a project.

Usage

```
fetchPermissions(project, url = restUrl(), config = NULL)
```

Arguments

`project`	String containing the project name.
`url`	String containing the URL of the gypsum REST API.
`config`	Deprecated and ignored.

Value

List containing the permissions for this project. This has the following elements:

- `owners`, a character vector containing the GitHub users or organizations that are owners of this project.
- `uploaders`, a list of lists specifying the users or organizations who are authorzied to upload to this project. Each entry is a list with the following fields:
  - `id`, a string containing the GitHub user or organization that is authorized to upload.
  - (optional) `asset`, a string containing the name of the asset that the uploader is allowed to upload to. If not provided, there is no restriction on the uploaded asset name.
  - (optional) `version`, a string containing the name of the version that the uploader is allowed to upload to. If not provided, there is no restriction on the uploaded version name.
  - (optional) `until`, a POSIXct object containing the expiry date of this authorization. If not provided, the authorization does not expire.
  - (optional) `trusted`, whether the uploader is trusted. If not provided, defaults to `FALSE`.

Author(s)

Aaron Lun

See Also

`setPermissions`, to set the permissions.

Examples

```
fetchPermissions("test-R")
```

fetchQuota *Fetch project quota details*

Description

Fetch the quota details for a project.

Usage

```
fetchQuota(project, url = restUrl(), config = NULL)
```

Arguments

project	String containing the project name.
url	String containing the URL of the gypsum REST API.
config	Deprecated and ignored.

Value

List containing baseline, the baseline quota at time zero in bytes; growth_rate, the annual growth rate for the quota in bytes; and year, the creation year (i.e., time zero) for this project.

Author(s)

Aaron Lun

See Also

setQuota, to set the quota details.

Examples

```
fetchQuota("test-R")
```

fetchSummary *Fetch version summary*

Description

Fetch the summary for a version of an asset of a project.

Usage

```
fetchSummary(
  project,
  asset,
  version,
  cache = cacheDirectory(),
  overwrite = FALSE,
  url = restUrl(),
  config = NULL
)
```

Arguments

project	String containing the project name.
asset	String containing the asset name.
version	String containing the version name.
cache	String containing the cache directory. If NULL, no caching is performed.
overwrite	Logical scalar indicating whether to overwrite an existing file in cache, if one is present.
url	String containing the URL of the gypsum REST API.
config	Deprecated and ignored.

Value

List containing the summary for this version, with the following fields:

- upload_user_id, string containing the identity of the uploader.
- upload_start, a POSIXct object containing the upload start time.
- upload_finish, a POSIXct object containing the upload finish time.
- on_probation (optional), a logical scalar indicating whether the upload is probational. If missing, this can be assumed to be FALSE.

Author(s)

Aaron Lun

Examples

```
fetchSummary("test-R", "basic", "v1")
```

fetchUsage *Fetch project usage details*

Description

Fetch the quota usage for a project.

Usage

```
fetchUsage(project, url = restUrl(), config = NULL)
```

Arguments

project	String containing the project name.
url	String containing the URL of the gypsum REST API.
config	Deprecated and ignored.

Value

Numeric scalar specifying the quota usage for the project, in bytes.

Author(s)

Aaron Lun

See Also

refreshUsage, to recompute the used quota.

Examples

```
fetchUsage("test-R")
```

formatObjectMetadata *Format object-related metadata*

Description

Create object-related metadata to validate against the default schema from fetchMetadataSchema. This is intended for downstream package developers who are auto-generating metadata documents to be validated by validateMetadata.

Usage

```
formatObjectMetadata(x)
```

Arguments

x	An R object, typically an instance of a Bioconductor class.

Value

List containing the object-related metadata, typically stored in the `applications.takane` field of the metadata.

Author(s)

Aaron Lun

Examples

```
df <- S4Vectors::DataFrame(alpha=LETTERS, numeric=runif(26))
formatObjectMetadata(df)
```

`listAssets` *List assets*

Description

List all assets in a project.

Usage

```
listAssets(project, url = restUrl(), config = NULL)
```

Arguments

`project`	String containing the project name.
`url`	String containing the URL of the gypsum REST API.
`config`	Deprecated and ignored.

Value

Character vector of asset names.

Author(s)

Aaron Lun

Examples

```
listAssets("test-R")
```

`listFiles` *List files for a version*

Description

List files belonging to a version of a project asset.

Usage

```
listFiles(
  project,
  asset,
  version,
  prefix = NULL,
  include.. = TRUE,
  url = restUrl(),
  config = NULL
)
```

Arguments

`project`	String containing the project name.
`asset`	String containing the asset name.
`version`	String containing the version name.
`prefix`	String containing the remaining prefix for the object key. If provided, a file is only listed if its object key starts with `{project}/{asset}/{version}/{prefix}`. If `NULL`, all files associated with this version of the asset are listed.
`include..`	Logical scalar indicating whether to list files with `/..` in their object keys.
`url`	String containing the URL of the gypsum REST API.
`config`	Deprecated and ignored.

Value

Character vector of relative paths of files associated with the versioned asset.

Author(s)

Aaron Lun

Examples

```
listFiles("test-R", "basic", "v1")
```

listProjects *List all projects*

Description

List all projects in the gypsum backent.

Usage

```
listProjects(url = restUrl(), config = NULL)
```

Arguments

url	String containing the URL of the gypsum REST API.
config	Deprecated and ignored.

Value

Character vector of project names.

Author(s)

Aaron Lun

Examples

```
if (interactive()) {
    listProjects()
}
```

listVersions *List asset versions*

Description

List all versions of a project asset.

Usage

```
listVersions(project, asset, url = restUrl(), config = NULL)
```

Arguments

project	String containing the project name.
asset	String containing the asset name.
url	String containing the URL of the gypsum REST API.
config	Deprecated and ignored.

Value

Character vector of versions.

Author(s)

Aaron Lun

Examples

```
listVersions("test-R", "basic")
```

`prepareDirectoryUpload` *Prepare to upload a directory*

Description

Prepare to upload a directory's contents via `startUpload`. This goes through the directory to list its contents and convert symlinks to upload links.

Usage

```
prepareDirectoryUpload(
  directory,
  links = c("auto", "always", "never"),
  cache = cacheDirectory()
)
```

Arguments

`directory`	String containing the path to a directory, the contents of which are to be uploaded via `startUpload`.
`links`	String indicating how to handle symlinks in `directory`. • `"auto"` will attempt to convert symlinks into upload links. If the conversion fails, a regular upload is performed. • `"always"` will attempt to convert symlinks into upload links. If the conversion fails, an error is raised. • `"never"` will never attempt to convert symlinks into upload links. All symlinked files are treated as regular uploads.
`cache`	String containing a path to the cache directory, used to convert symlinks into upload links.

Details

Files in `directory` (that are not symlinks) are used as regular uploads, i.e., `files=` in `startUpload`.

If `directory` contains a symlink to a file in `cache`, we assume that it points to a file that was previously downloaded by, e.g., `saveFile` or `saveVersion`. Thus, instead of performing a regular upload, we attempt to create an upload link, i.e., `links=` in `startUpload`. This is achieved by examining the destination path of the symlink and inferring the link destination in the backend. Note that this still works if the symlinks are dangling.

If a symlink cannot be converted into an upload link, it will be used as a regular upload, i.e., the contents of the symlink destination will be uploaded by `startUpload`. In this case, an error will be raised if the symlink is dangling as there is no file that can actually be uploaded. If `links="always"`, an error is raised instead upon symlink conversion failure.

This function is intended to be used with `cloneVersion`, which creates symlinks to files in `cache`.

Value

List containing `files`, a character vector to be used as `files=` in `startUpload`; and `links`, a data frame to be used as `links=` in `startUpload`.

See Also

`startUpload`, to actually start the upload.

`cloneVersion`, to prepare the symlinks.

Examples

```
tmp <- tempfile()
out <- cloneVersion("test-R", "basic", "v1", destination=tmp)
write(file=file.path(tmp, "heanna"), "sumire")
prepareDirectoryUpload(tmp)
```

---

publicS3Config *Public S3 configuration*

Description

Fetch S3 credentials and other configuration details for read-only access to the underlying gypsum bucket.

Usage

```
publicS3Config(refresh = FALSE, url = restUrl(), cache = cacheDirectory())
```

Arguments

`refresh`	Logical scalar indicating whether to refresh the credentials in the in-memory cache.
`url`	String containing a URL to the gypsum REST API.
`cache`	String containing a path to the cache directory, to store the configuration across R sessions.

Details

The configuration is obtained through a query to `url` on the first use of this function. The result is automatically cached in memory and on disk to reduce the number of network requests to the API. New credentials are automatically fetched if the on-disk cache is older than a week; this refresh can be performed manually by calling this function with `refresh=TRUE`.

Value

List containing:

- `key`, a string containing the read-only S3 access key ID.
- `secret`, a string containing the associated S3 access secret.
- `bucket`, a string containing the name of the bucket.
- `endpoint`, a string containing the URL for the S3 API.

Author(s)

Aaron Lun

Examples

```
publicS3Config()
```

`refreshLatest` *Refresh the latest version*

Description

Recompute the latest version of a project's asset. This is useful on rare occasions where multiple simultaneous uploads cause the latest version to be slightly out of sync.

Usage

```
refreshLatest(project, asset, url = restUrl(), token = accessToken())
```

Arguments

`project`	String containing the project name.
`asset`	String containing the asset name.
`url`	String containing the URL of the gypsum REST API.
`token`	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to a gypsum administrator account.

Value

String containing the latest version of the project, or `NULL` if there are no non-probational versions.

Author(s)

Aaron Lun

See Also

fetchLatest, to get the latest version without recomputing it.

Examples

```
if (interactive()) {
    refreshLatest("test-R", "basic")
}
```

`refreshUsage` *Refresh the quota usage*

Description

Recompute the quota usage of a project. This is useful on rare occasions where multiple simultaneous uploads cause the usage calculations to be out of sync.

Usage

```
refreshUsage(project, url = restUrl(), token = accessToken())
```

Arguments

`project`	String containing the project name.
`url`	String containing the URL of the gypsum REST API.
`token`	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to a gypsum administrator account.

Value

Numeric scalar specifying the total quota usage of this project, in bytes.

Author(s)

Aaron Lun

See Also

fetchUsage, to get the usage without recomputing it.

Examples

```
if (interactive()) {
    refreshUsage("test-R")
}
```

`rejectProbation` *Reject a probational upload*

Description

Pretty much as it says: reject a probational upload of a version of a project's asset. This removes all files associated with that version.

Usage

```
rejectProbation(
  project,
  asset,
  version,
  url = restUrl(),
  token = accessToken()
)
```

Arguments

`project`	String containing the project name.
`asset`	String containing the asset name.
`version`	String containing the version name.
`url`	String containing the URL of the gypsum REST API.
`token`	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to an owner of `project`.

Value

`NULL` is invisibly returned upon successful rejection.

Author(s)

Aaron Lun

See Also

`approveProbation`, to approve the probational upload.

`startUpload`, to specify probational uploads.

Examples

```
if (interactive()) {
    # Mocking up a versioned asset.
    init <- startUpload(
        project="test-R",
        asset="probation-reject",
        version="v1",
        files=character(0),
        probation=TRUE
    )
```

```
    completeUpload(init)

    # Rejecting the probation:
    rejectProbation("test-R", "probation-reject", "v1")
}
```

`removeAsset` *Remove an asset*

Description

Remove an asset of a project from the gypsum backend.

Usage

```
removeAsset(project, asset, url = restUrl(), token = accessToken())
```

Arguments

`project`	String containing the project to remove.
`asset`	String containing the asset to remove.
`url`	String containing the URL of the gypsum REST API.
`token`	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to a gypsum administrator account.

Value

`NULL` is invisibly returned if the asset was successfully removed.

Author(s)

Aaron Lun

See Also

`removeProject`, to remove a project.

`removeVersion`, to remove a specific version.

Examples

```
if (interactive()) {
    # Mocking up a versioned asset.
    init <- startUpload(
        project="test-R",
        asset="removal",
        version="v1",
        files=character(0),
        probation=TRUE
    )
    completeUpload(init)

    removeAsset("test-R", asset="removal")
}
```

removeProject *Remove a project*

Description

Remove a project from the gypsum backend.

Usage

```
removeProject(project, url = restUrl(), token = accessToken())
```

Arguments

project	String containing the project to remove.
url	String containing the URL of the gypsum REST API.
token	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to a gypsum administrator account.

Value

NULL is invisibly returned if the project was successfully removed.

Author(s)

Aaron Lun

See Also

createProject, to create a project.

removeAsset and removeVersion, to remove an asset or version.

Examples

```
if (interactive()) {
    createProject("test-R-remove", owners="LTLA")
    removeProject("test-R-remove")
}
```

removeVersion *Remove a version of an asset*

Description

Remove a version of an asset from the gypsum backend.

Usage

```
removeVersion(project, asset, version, url = restUrl(), token = accessToken())
```

Arguments

project	String containing the project to remove.
asset	String containing the asset to remove.
version	String containing the version of the asset to remove.
url	String containing the URL of the gypsum REST API.
token	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to a gypsum administrator account.

Value

NULL is invisibly returned if the project or its contents was successfully removed.

Author(s)

Aaron Lun

See Also

removeAsset and removeProject, to remove an asset or project.

Examples

```
if (interactive()) {
    # Mocking up a versioned asset.
    init <- startUpload(
        project="test-R",
        asset="removal",
        version="v1",
        files=character(0),
        probation=TRUE
    )
    completeUpload(init)

    removeVersion("test-R", asset="removal", version="v1")
}
```

---

resolveLinks *Resolve links in the cache directory*

Description

Create hard links (or copies, if filesystem links are not supported) for linked-from files to their link destinations.

Usage

```
resolveLinks(
  project,
  asset,
  version,
  cache = cacheDirectory(),
  overwrite = FALSE,
  url = restUrl(),
  config = NULL
)
```

Arguments

project	String containing the project name.
asset	String containing the asset name.
version	String containing the version name.
cache	String containing the path to the cache directory.
overwrite	Logical scalar indicating whether to replace existing files at the linked-from paths.
url	String containing the URL of the gypsum REST API.
config	Deprecated and ignored.

Value

NULL is returned on successful completion.

Author(s)

Aaron Lun

Examples

```
cache <- tempfile()
saveVersion("test-R", "basic", "v3", relink=FALSE, cache=cache)
list.files(cache, recursive=TRUE, all.files=TRUE)

resolveLinks("test-R", "basic", "v3", cache=cache)
list.files(cache, recursive=TRUE, all.files=TRUE)
```

---

restUrl *URL for the REST API*

Description

Get or set the URL for the gypsum REST API.

Usage

```
restUrl(url)
```

Arguments

url	String containing the URL of the REST API.

Value

If url is missing, the current setting of the URL is returned.

If url is provided, it is used to replace the current setting of the URL, and the *previous* setting of the URL is invisibly returned.

Author(s)

Aaron Lun

Examples

```
restUrl()
old <- restUrl("https://some-other.rest-api.io") # replace it.
restUrl()
restUrl(old) # setting it back.
```

---

saveFile *Save a file from a version of a project asset*

Description

Download a file from the gypsum bucket, for a version of an asset of a project.

Usage

```
saveFile(
  project,
  asset,
  version,
  path,
  cache = cacheDirectory(),
  overwrite = FALSE,
  url = restUrl(),
  config = NULL
)
```

Arguments

project	String containing the project name.
asset	String containing the asset name.
version	String containing the version name.
path	String containing the suffix of the object key for the file of interest, i.e., the relative "path" inside the version's "subdirectory". The full object key is defined as {project}/{asset}/{version}/{path}.
cache	String containing the path to the cache directory.

overwrite	Logical scalar indicating whether to overwrite an existing file in cache. If FALSE and the file exists in cache, the download is skipped.
url	String containing the URL of the gypsum REST API.
config	Deprecated and ignored.

Details

The full object key is defined as {project}/{asset}/{version}/{path}. If no file exists in the project-asset-version combination at path, this function will check the ..links file to check whether path refers to a linked-from file. If so, the contents of the link destination is downloaded to the cache and a link/copy is created at the returned file path.

Value

The file is downloaded to the local file system. The destination file path is returned.

Author(s)

Aaron Lun

See Also

saveVersion, to save all files with the same prefix.

cacheDirectory, for file caching.

Examples

```
out <- saveFile("test-R", "basic", "v1", "blah.txt")
readLines(out)
```

---

saveVersion *Save all files for a version of a project asset*

Description

Download all files associated with a version of an asset of a project from the gypsum bucket.

Usage

```
saveVersion(
  project,
  asset,
  version,
  cache = cacheDirectory(),
  overwrite = FALSE,
  relink = TRUE,
  concurrent = 1,
  url = restUrl(),
  config = NULL
)
```

Arguments

project	String containing the project name.
asset	String containing the asset name.
version	String containing the version name.
cache	String containing the path to the cache directory.
overwrite	Logical scalar indicating whether to overwrite existing files in the cache. If FALSE and the files already exist in cache, the download is skipped.
relink	Logical scalar indicating whether links should be resolved, see resolveLinks.
concurrent	Integer specifying the number of concurrent downloads.
url	String containing the URL of the gypsum REST API.
config	Deprecated and ignored.

Value

The version's files are downloaded to the local file system, and the path to the local subdirectory is returned.

Author(s)

Aaron Lun

See Also

saveFile, to save a single file.

cacheDirectory, for file caching.

Examples

```
out <- saveVersion("test-R", "basic", "v1")
list.files(out, recursive=TRUE, all.files=TRUE)
```

---

searchMetadata *Search the metadata database*

Description

Search a SQLite database containing metadata from the gypsum backend. This is based on a pre-computed tokenization of all string properties in each metadata document; see https://github.com/ArtifactDB/bioconductor-metadata-index for details.

Usage

```
searchMetadata(path, query, latest = TRUE, include.metadata = TRUE)

gsc(
  text = NULL,
  project = NULL,
  asset = NULL,
  version = NULL,
  path = NULL,
  user = NULL,
  time = NULL,
  field = NULL,
  partial = FALSE,
  after = TRUE
)

searchMetadataFilter(
  query,
  pid.name = "paths.pid",
  project.name = "versions.project",
  asset.name = "versions.asset",
  version.name = "versions.version",
  path.name = "paths.path",
  user.name = "versions.user",
  time.name = "versions.time"
)
```

Arguments

path	For searchMetadata, a string containing a path to a SQLite file, usually obtained via fetchMetadataDatabase. For gsc, the suffix of the object key of the metadata document, i.e., the relative "path" to the metadata file inside the version's "directory". This may be missing as long as other arguments are supplied to gsc.
query	A gypsum.search.clause object, typically produced by gsc or translateTextQuery.
latest	Logical scalar indicating whether to only search for matches within the latest version of each asset.
include.metadata	Logical scalar indicating whether metadata should be returned.
text	String containing the text to query on. This will be automatically tokenized, see Details. This may be missing as long as other arguments are supplied to gsc.
project	String containing the name of the project. This may be missing as long as other arguments are supplied to gsc.
asset	String containing the name of the asset. This may be missing as long as other arguments are supplied to gsc.
version	String containing the name of the version. This may be missing as long as other arguments are supplied to gsc.
user	String containing the user ID of the uploader. This may be missing as long as other arguments are supplied to gsc.

time	Number specifying the Unix timestamp (in seconds) at which the upload was finished. This may be missing as long as other arguments are supplied to gsc.
field	String specifying the name of the metadata field in which to search for text. If NULL, the search is performed on all available metadata fields.
partial	For gsc, a logical scalar indicating whether text, project, asset, version, path or user contains SQLite wildcards (%, _) for a partial search. For text, setting partial=TRUE also ensures that the wildcards are preserved during tokenization.
after	Logical scalar indicating whether to search for documents that were uploaded after time. If FALSE, the search will instead consider documents that were uploaded at or before time.
pid.name	String containing the name/alias of the column of the paths table that contains the path ID.
project.name	String containing the name/alias of the column of the versions table that contains the project name.
asset.name	String containing the name/alias of the column of the versions table that contains the asset name.
version.name	String containing the name/alias of the column of the versions table that contains the version name.
path.name	String containing the name/alias of the column of the paths table that contains the path name.
user.name	String containing the name/alias of the column of the versions table that contains the user ID of the uploader.
time.name	String containing the name/alias of the column of the versions table that contains the upload time.

Details

Each query string is tokenized by converting it to lower case and splitting it on characters that are not Unicode letters/numbers or a dash. We currently do not remove diacritics so these will need to be converted to ASCII by the user. If a text query involves only non-letter/number/dash characters, the filter will not be well-defined and will be ignored when constructing SQL statements. The metadata document/field is only considered to match the query string if all of the tokens can be found in that document/field (in any order).

Value

For searchMetadata, a data frame specifying the contaning the search results.

- The project, asset and version columns specify the version of the project asset with matching metadata.
- The path column contains the suffix of the object key of the metadata document, i.e., the relative "path" within the version's "directory" to the metadata document. The full object key of the document inside the bucket is defined as {project}/{asset}/{version}/{path}.
- The user column contains the identity of the uploading user.
- The time column contains the time of the upload.
- If include.metadata=TRUE, a metadata column is present with the nested metadata for each match.

- If `latest=TRUE`, a `latest` column is present indicating whether the matching version is the latest for its asset. Otherwise, only the latest version is returned.

For `searchMetadataFilter`, a list containing `where`, a string can be directly used as a WHERE filter condition in a SQL SELECT statement; and `parameters`, the parameter bindings to be used in `where`. The return value may also be `NULL` if the query has no well-defined filter.

For `gsc`, a gypsum.search.clause object that can be used in `|`, `&` and `!` to create more complex queries involving multiple clauses.

Author(s)

Aaron Lun

See Also

`fetchMetadataDatabase`, to download and cache the database files.

https://github.com/ArtifactDB/bioconductor-metadata-index, for details on the SQLite file contents and table structure.

`translateTextQuery`, to create a gypsum.search.clause from human-friendly syntax.

Examples

```
path <- fetchMetadataDatabase()
searchMetadata(path, gsc("mouse brain"), include.metadata=FALSE)

# Now for a slightly more complex query:
query <- (gsc("brain") | gsc("pancreas")) & gsc("10090", field="taxonomy_id")
searchMetadata(path, query, include.metadata=FALSE)

# Throwing in some wildcards.
has.neuro <- gsc("Neuro%", partial=TRUE)
searchMetadata(path, has.neuro, include.metadata=FALSE)

# We can also query other properties.
datasets <- gsc(project="scRNAseq") & gsc(asset="l%", partial=TRUE)
searchMetadata(path, datasets, include.metadata=FALSE)
```

---

setPermissions *Set project permissions*

Description

Set the owner and uploader permissions for a project.

Usage

```
setPermissions(
  project,
  owners = NULL,
  uploaders = NULL,
  append = TRUE,
```

```
  url = restUrl(),
  token = accessToken()
)
```

Arguments

project	String containing the project name.
owners	Character vector containing the GitHub users or organizations that are owners of this project. If NULL, no change is made to the existing owners of the project.
uploaders	List specifying the authorized uploaders for this project. See the uploaders field in the fetchPermissions return value for the expected format. If NULL, no change is made to the existing uploaders of the project.
append	Logical scalar indicating whether owners and uploaders should be appended to the existing owners and uploaders, respectively, of the project. If FALSE, the owners and uploaders are used to replace the existing values.
url	String containing the URL of the gypsum REST API.
token	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to an owner of the project.

Value

NULL is invisibly returned upon successful setting of the permissions.

Author(s)

Aaron Lun

See Also

fetchPermissions, to fetch the permissions.

Examples

```
if (interactive()) {
    # Creating a project for demonstration purposes.
    createProject("test-R-perms", owners="LTLA")

    # Setting extra permissions on this project.
    setPermissions("test-R-perms",
        owners="jkanche",
        uploaders=list(list(id="lawremi", until=Sys.time() + 1000))
    )
}
```

setQuota *Set project quota*

Description

Set the storage quota for a project.

Usage

```
setQuota(
  project,
  baseline = NULL,
  growth = NULL,
  year = NULL,
  url = restUrl(),
  token = accessToken()
)
```

Arguments

project	String containing the project name.
baseline	Numeric scalar specifying the baseline quota (i.e., at time zero) in bytes. If NULL, no change is made to the existing baseline of the project.
growth	Numeric scalar specifying the annual growth rate of the quota, in bytes. If NULL, no change is made to the existing growth rate of the project.
year	Integer scalar specifying the year of creation (i.e., time zero) for the project. If NULL, no change is made to the existing creation year of the project.
url	String containing the URL of the gypsum REST API.
token	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to a gypsum administrator account.

Value

NULL is invisibly returned upon successful setting of the quota.

Author(s)

Aaron Lun

See Also

fetchQuota, to fetch the quota.

Examples

```
if (interactive()) {
    # Creating a project for demonstration purposes.
    createProject("test-R-quota", owners="LTLA")

    # Setting a baseline of 10 GB with 5 GB in growth per year.
    setQuota("test-R-quota", baseline=10^10, growth=5^9, year=2019)
```

```
}
```

`startUpload` *Start an upload*

Description

Start an upload of a new version of an asset, or a new asset of a project.

Usage

```
startUpload(
  project,
  asset,
  version,
  files,
  links = NULL,
  deduplicate = TRUE,
  probation = FALSE,
  url = restUrl(),
  token = accessToken(),
  directory = NULL
)
```

Arguments

`project`	String containing the project name.
`asset`	String containing the asset name. This should not contain `/` or start with `...`
`version`	String containing the version name. This should not contain `/` or start with `...`
`files`	Character vector containing the paths of the files to be uploaded. These should be relative to the version's `directory`. Alternatively, a data frame where each row corresponds to a file and contains information about those files. This data frame should contain the following fields: • `path`, a string containing the relative path of the file inside the version's subdirectory. • `size`, a non-negative integer specifying the size of the file in bytes. • `md5sum`, a string containing the hex-encoded MD5 checksum of the file. • (optional) `dedup`, a logical indicating whether deduplication should be attempted for each file.
`links`	A data frame where each row corresponds to a linked-from file and contains the link destination for that file. This data frame should contain the following fields: • `from.path`, a string containing the relative path of the file inside the version's subdirectory. • `to.project`, a string containing the project of the list destination. • `to.asset`, a string containing the asset of the list destination. • `to.version`, a string containing the version of the list destination.

- `to.path`, a string containing the path of the list destination.

`deduplicate` Logical scalar indicating whether the backend should attempt deduplication of `files` in the immediately previous version. Only has an effect if `files` is not a data frame or if the dedup field is missing.

`probation` Logical scalar indicating whether to perform a probational upload. Such uploads must be approved by the project owner before they are considered official.

`url` String containing the URL of the gypsum REST API.

`token` String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to a user that is authorized to upload to the specified `project`.

`directory` String containing the path to a directory containing the `files` to be uploaded. This directory is assumed to correspond to a version of an asset. It only has an effect if `files` is a character vector, as it is used to determine the MD5 checksums and sizes. If `NULL`, `directory` is set to the current working directory.

Value

List containing:

- `file_urls`, a list of lists containing information about each file to be uploaded. This is used by `uploadFiles`.
- `complete_url`, a string containing the completion URL, to be used by `completeUpload`.
- `abort_url`, a string specifying the abort URL, to be used by `abortUpload`.
- `session_token`, a string for authenticating to the newly initialized upload session.

Author(s)

Aaron Lun

See Also

`uploadFiles`, to actually upload the files.

`completeUpload`, to indicate that the upload is completed.

`abortUpload`, to abort an upload in progress.

`prepareDirectoryUpload`, to create `files` and `links` from a directory.

Examples

```
tmp <- tempfile()
dir.create(tmp)
write(file=file.path(tmp, "blah.txt"), LETTERS)
dir.create(file.path(tmp, "foo"))
write(file=file.path(tmp, "foo", "bar.txt"), 1:10)

if (interactive()) {
    blob <- startUpload(
        project="test-R",
        asset="upload-start-check",
        version="v1",
        files=list.files(tmp, recursive=TRUE),
        directory=tmp
```

```
    )
    print(blob)

    abortUpload(blob) # just cleaning up after we're done.
}
```

`translateTextQuery` *Translate a plain-text query to a gypsum.search.clause*

Description

Translate a plain-text query in user-friendly format to a gypsum.search.clause, equivalent to that created by `gsc` with the corresponding boolean operations.

Usage

```
translateTextQuery(query)
```

Arguments

`query`	String containing a query in user-friendly format to a gypsum.search.clause.

Details

For text searches, we support a more human-readable syntax for boolean operations in the query. The search string below will look for all metadata documents that match `foo` or `bar` but not `whee`:

```
(foo OR bar) AND NOT whee
```

The `AND`, `OR` and `NOT` (note the all-caps!) are automatically translated to the corresponding search clauses. This can be combined with parentheses to control precedence; otherwise, `AND` takes precedence over `OR`, and `NOT` takes precedence over both. Note that any sequence of adjacent text terms are implicitly AND'd together, so the two expressions below are equivalent:

```
foo bar whee
foo AND bar AND whee
```

Users can prefix any sequence of text terms with the name of a metadata field, to only search for matches within that field of the metadata file. For example:

```
(title: prostate cancer) AND (genome: GRCh38 OR genome: GRCm38)
```

Note that this does not extend to the `AND`, `OR` and `NOT` keywords, e.g., `title:foo OR bar` will not limit the search for `bar` to the `title` field.

If a `%` wildcard is present in a search term, its corresponding text search clause is configured to perform a partial search.

Value

A gypsum.search.clause representing `query`.

Author(s)

Aaron Lun

Examples

```
str(translateTextQuery("foo AND bar"))
str(translateTextQuery("foo AND bar%"))
str(translateTextQuery("(foo AND NOT bar) OR whee"))
str(translateTextQuery("blah:foo whee AND stuff: bar blob"))
```

---

unlockProject *Unlock a project*

Description

Unlock a project on the gypsum backend. This is typically caused by a partial upload that was not properly aborted.

Usage

```
unlockProject(project, url = restUrl(), token = accessToken())
```

Arguments

project	String containing the project to remove.
url	String containing the URL of the gypsum REST API.
token	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to a gypsum administrator account.

Value

NULL is invisibly returned if the project was successfully removed.

Author(s)

Aaron Lun

See Also

abortUpload, to correctly abort an upload upon failure.

Examples

```
if (interactive()) {
    unlockProject("test-R")
}
```

`uploadDirectory` *Upload a directory to the gypsum backend*

Description

Convenience method to upload a directory to the gypsum backend as a versioned asset of a project. This requires uploader permissions to the relevant project.

Usage

```
uploadDirectory(
  directory,
  project,
  asset,
  version,
  cache = cacheDirectory(),
  deduplicate = TRUE,
  probation = FALSE,
  url = restUrl(),
  token = accessToken(),
  concurrent = 1,
  abort.failed = TRUE
)
```

Arguments

`directory`	String containing the path to a directory to be uploaded.
`project`	String containing the project name.
`asset`	String containing the asset name. This should not contain `/` or start with `...`
`version`	String containing the version name. This should not contain `/` or start with `...`
`cache`	String containing the path to the cache for saving files, e.g., in `saveVersion`. Used to convert symbolic links to upload links, see `prepareDirectoryUpload`.
`deduplicate`	Logical scalar indicating whether the backend should attempt deduplication of `files` in the immediately previous version. Only has an effect if `files` is not a data frame or if the dedup field is missing.
`probation`	Logical scalar indicating whether to perform a probational upload. Such uploads must be approved by the project owner before they are considered official.
`url`	String containing the URL of the gypsum REST API.
`token`	String containing a GitHub access token to authenticate to the gypsum REST API. The token must refer to a user that is authorized to upload to the specified `project`.
`concurrent`	Integer scalar specifying the number of concurrent uploads in `uploadFiles`.
`abort.failed`	Logical scalar indicating whether to abort the upload on any failure. Setting this to `FALSE` can be helpful for diagnosing upload problems.

Details

This function is a wrapper around `prepareDirectoryUpload` and `startUpload` and friends. The aim is to streamline the upload of a directory's contents when no customization of the file listing is required.

Value

On successful upload, NULL is invisibly returned.

Author(s)

Aaron Lun

Examples

```
tmp <- tempfile()
dir.create(tmp)
write(file=file.path(tmp, "blah.txt"), LETTERS)
dir.create(file.path(tmp, "foo"))
write(file=file.path(tmp, "foo", "bar.txt"), 1:10)

if (interactive()) {
    # Uploading a probational version for test purposes.
    uploadDirectory(staging, "test-R", "upload-dir-check", version, probation=TRUE)

    # Cleaning up after ourselves.
    gypsum::rejectProbation("test-R", "upload-dir-check", version)
}
```

---

uploadFiles *Upload files for a versioned asset*

Description

Upload files in an initialized upload session for a version of an asset.

Usage

```
uploadFiles(init, directory = NULL, url = restUrl(), concurrent = 1)
```

Arguments

init	List containing file_urls and session_token. This is typically the return value from startUpload.
directory	String containing the path to a directory containing the files to be uploaded. This directory is assumed to correspond to a version of an asset. It only has an effect if files is a character vector, as it is used to determine the MD5 checksums and sizes. If NULL, directory is set to the current working directory.
url	String containing the URL of the gypsum REST API.
concurrent	Integer specifying the number of concurrent uploads.

Value

NULL is invisibly returned on successful upload of all files.

Author(s)

Aaron Lun

See Also

startUpload, to create init.

Examples

```
tmp <- tempfile()
dir.create(tmp)
write(file=file.path(tmp, "blah.txt"), LETTERS)
dir.create(file.path(tmp, "foo"))
write(file=file.path(tmp, "foo", "bar.txt"), 1:10)

if (interactive()) {
    init <- startUpload(
        project="test-R",
        asset="upload-files-check",
        version="v1",
        files=list.files(tmp, recursive=TRUE),
        directory=tmp
    )

    # Executing the upload for all files.
    uploadFiles(init, directory=tmp)

    # Cleaning up after we're done.
    abortUpload(init)
}
```

---

validateMetadata *Validate metadata against a JSON schema*

Description

Validate metadata against a JSON schema for a SQLite database. This ensures that it can be successfully inserted in the database in downstream indexing steps.

Usage

```
validateMetadata(metadata, schema = fetchMetadataSchema(), stringify = NULL)
```

Arguments

metadata	Metadata to be checked. This is usually an R object like a named list, but may also be a JSON-formatted string.
schema	String containing a path to a schema.
stringify	Logical scalar indicating whether to convert metadata to a JSON-formatted string. Defaults to TRUE if metadata is not already a string.

Value

NULL is invisibly returned upon successful validation.

Author(s)

Aaron Lun

See Also

fetchMetadataSchema, to get the JSON schemas.

fetchMetadataDatabase, to obtain the SQLite database files.

Examples

```
metadata <- list(
    title="Fatherhood",
    description="Luke ich bin dein Vater.",
    sources=list(
       list(provider="GEO", id="GSE12345")
    ),
    taxonomy_id=list("9606"),
    genome=list("GRCm38"),
    maintainer_name="Darth Vader",
    maintainer_email="vader@empire.gov",
    bioconductor_version="3.10"
)

validateMetadata(metadata)
```

Index