Contents

abortUpload ......................................................... 3
accessToken ......................................................... 4
approveProbation ............................................... 5
cacheDirectory .................................................... 6
clonVersion ....................................................... 7
completeUpload .................................................. 9
createProject .................................................... 10
fetchLatest ......................................................... 11
fetchManifest ..................................................... 12
fetchMetadataDatabase .......................................... 13
fetchMetadataSchema ............................................. 14
fetchPermissions ................................................. 15
fetchQuota ......................................................... 16
fetchSummary ...................................................... 17
fetchUsage ......................................................... 18
formatObjectMetadata ............................................ 19
listAssets ......................................................... 20
listFiles .......................................................... 20
listProjects ....................................................... 21
listVersions ....................................................... 22
prepareDirectoryUpload ......................................... 23
publicS3Config .................................................... 24
refreshLatest ...................................................... 25
refreshUsage ....................................................... 26
rejectProbation .................................................... 27
removeAsset ....................................................... 28
removeProject ..................................................... 29
removeVersion ..................................................... 30
resolveLinks ....................................................... 31
restUrl ............................................................ 32
saveFile ........................................................... 33
saveVersion ....................................................... 34
searchMetadataText .............................................. 35
setPermissions .................................................... 37
setQuota .......................................................... 38
startUpload ....................................................... 39
uploadDirectory .................................................. 41
uploadFiles ....................................................... 43
validateMetadata ................................................. 44

Index ............................................................... 46
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

```r
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

```r
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 replace the current setting of the cache directory, and the previous setting is invisibly returned.

Author(s)

Aaron Lun

Examples

```r
cacheDirectory()
old <- cacheDirectory(tempfile())

# setting it back.
```
cloneVersion

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.
- **config**: Configuration object for the S3 bucket, see publicS3Config for details.
- **...**: 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.
completeUpload

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.
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.
fetchLatest

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, config = publicS3Config())`
fetchManifest

**Arguments**

- `project` String containing the project name.
- `asset` String containing the asset name.
- `config` Configuration object for the S3 bucket, see `publicS3Config` for details.

**Value**

String containing the latest version of the project.

**Author(s)**

Aaron Lun

**See Also**

`refreshLatest`, to refresh the latest version.

**Examples**

```r
collectAssets("test-R", "basic")
```

---

**Description**

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

**Usage**

```r
fetchManifest(  
  project,  
  asset,  
  version,  
  cache = cacheDirectory(),  
  overwrite = FALSE,  
  config = publicS3Config(cache = cache)
)
```
fetchMetadataDatabase

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.
config Configuration object for the S3 bucket, see publicS3Config for details.

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

```r
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

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

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, config = publicS3Config())

Arguments

project String containing the project name.

config Configuration object for the S3 bucket, see publicS3Config for details.
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 authorized 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, config = publicS3Config())

Arguments

- project
  - String containing the project name.
- config
  - Configuration object for the S3 bucket, see publicS3Config for details.
**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**

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

---

**Description**

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

**Usage**

```r
fetchSummary(
  project,  
  asset,  
  version,  
  cache = cacheDirectory(),  
  overwrite = FALSE,  
  config = publicS3Config(cache = cache)
)
```

**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.
- **config**: Configuration object for the S3 bucket, see `publicS3Config` for details.
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(project, config = publicS3Config())

Description

Fetch the quota usage for a project.

Usage

fetchUsage(project, config = publicS3Config())

Arguments

project String containing the project name.
config Configuration object for the S3 bucket, see publicS3Config for details.

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**

```r
defetchUsage("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**

```r
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**

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

**List assets**

**Description**

List all assets in a project.

**Usage**

```
listAssets(project, config = publicS3Config())
```

**Arguments**

- `project`: String containing the project name.
- `config`: Configuration object for the S3 bucket, see `publicS3Config` for details.

**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,
  config = publicS3Config()
)
```
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.
- **config**: Configuration object for the S3 bucket, see `publicS3Config` for details.

Value

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

Author(s)

Aaron Lun

Examples

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

---

**listProjects**  

*List all projects*

Description

List all projects in the gypsum backent.

Usage

```r
listProjects(config = publicS3Config())
```

Arguments

- **config**: Configuration object for the S3 bucket, see `publicS3Config` for details.

Value

Character vector of project names.

Author(s)

Aaron Lun
Examples

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

---

**listVersions**  
*List asset versions*

**Description**

List all versions of a project asset.

**Usage**

```r
listVersions(project, asset, config = publicS3Config())
```

**Arguments**

- **project**: String containing the project name.
- **asset**: String containing the asset name.
- **config**: Configuration object for the S3 bucket, see `publicS3Config` for details.

**Value**

Character vector of versions.

**Author(s)**

Aaron Lun

**Examples**

```r
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

```r
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)
refreshLatest

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

```java
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

```r
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

```r
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

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

| 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**

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

**Arguments**

- `project` (String): String containing the project name.
- `asset` (String): String containing the asset name.
- `version` (String): String containing the version name.
- `url` (String): String containing the URL of the gypsum REST API.
- `token` (String): 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

```r
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**

```r
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

```r
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

```r
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`
- `removeProject`

### Examples

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

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

### Usage

```r
resolveLinks(
  project,
  asset,
  version,
  cache = cacheDirectory(),
  overwrite = FALSE,
  config = publicS3Config(cache = cache)
)
```

### 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.
- **config**: Configuration object for the S3 bucket, see `publicS3Config` for details.

### Value

NULL is returned on successful completion.

### Author(s)

Aaron Lun
Examples

```r
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**

```r
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 replace the current setting of the URL, and the *previous* setting of the URL is invisibly returned.

**Author(s)**

Aaron Lun

**Examples**

```r
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

```r
saveFile(
  project,
  asset,
  version,
  path,
  cache = cacheDirectory(),
  overwrite = FALSE,
  config = publicS3Config(cache = cache)
)
```

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.
- **config**: Configuration object for the S3 bucket, see `publicS3Config` for details.

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

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

---

### Description

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

### Usage

```r
saveVersion(
  project, asset, version,
  cache = cacheDirectory(),
  overwrite = FALSE,
  relink = TRUE,
  concurrent = 1,
  config = publicS3Config(cache = cache)
)
```

### 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.
- **config**: Configuration object for the S3 bucket, see `publicS3Config` for details.

### Value

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

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)

---

searchMetadataText  Text search on the metadata database

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

Usage
searchMetadataText(path, query, latest = TRUE, include.metadata = TRUE)

defineTextQuery(text, field = NULL, partial = FALSE)

searchMetadataTextFilter(query, pid.name = "paths.pid")

Arguments
path  String containing a path to a SQLite file, usually obtained via fetchMetadataDatabase.
query  Character vector specifying the query to execute. Alternatively, a gypsum.search.object produced by defineTextQuery.
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.
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  Logical scalar indicating whether text contains SQLite wildcards (\%, _) for a partial search. If TRUE, the wildcards are preserved during tokenization.
pid.name  String containing the name/alias of the column of the paths table that contains the path ID.
Details

Each 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.

For convenience, a non-empty character vector may be used in `query`. A character vector of length 1 is treated as shorthand for a text query with default arguments in `defineTextQuery`. A character vector of length greater than 1 is treated as shorthand for an AND operation on default text queries for each of the individual strings.

Value

For `searchMetadataText`, a data frame specifying the containing the search results.

- The `project`, `asset` and `version` columns contain the identity of the version 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}`.
- 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 `searchMetadataTextFilter`, 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 `defineTextQuery`, a `gypsum.search.clause` object that can be used in |, & and ! to create more complex queries involving multiple text 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.

Examples

```r
path <- fetchMetadataDatabase()
searchMetadataText(path, c("mouse", "brain"), include.metadata=FALSE)

# Now for a slightly more complex query:
is.mouse <- defineTextQuery("10090", field="taxonomy_id")
query <- (defineTextQuery("brain") | defineTextQuery("pancreas")) & is.mouse
```
setPermissions

searchMetadataText(path, query, include.metadata=FALSE)

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

```
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
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.

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))
  )
}
```
Value

NULL is invisibly returned upon successful setting of the quota.

Author(s)

Aaron Lun

See Also

fetchQuota, to fetch the quota.

Examples

```r
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)
}
```

Description

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

Usage

```r
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.

directories  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.
uploadDirectory

**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**

```r
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.
}
```

---

**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**

```r
uploadDirectory(
  directory,  # path to the directory
  project,   # project name
  asset,     # asset name
  version,   # version number
```
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Metadata to be checked. This is usually an R object like a named list, but may also be a JSON-formatted string.</td>
</tr>
<tr>
<td>schema</td>
<td>String containing a path to a schema.</td>
</tr>
<tr>
<td>stringify</td>
<td>Logical scalar indicating whether to convert <code>metadata</code> to a JSON-formatted string. Defaults to <code>TRUE</code> if <code>metadata</code> is not already a string.</td>
</tr>
</tbody>
</table>
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

```r
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

abortUpload, 3, 40, 41
accessToken, 4
approveProbation, 5, 27
cacheDirectory, 6, 7, 34, 35
cloneVersion, 7, 23, 24
completeUpload, 9, 40, 41
createProject, 10, 29
defineTextQuery (searchMetadataText), 35
fetchLatest, 11, 26
fetchManifest, 12
fetchMetadataDatabase, 13, 15, 35, 36, 45
fetchMetadataSchema, 14, 15, 19, 45
fetchPermissions, 11, 15, 37, 38
fetchQuota, 16, 39
fetchSummary, 17
fetchUsage, 18, 26
formatObjectMetadata, 19
gypsum.search.clause
  (searchMetadataText), 35
listAssets, 20
listFiles, 20
listProjects, 21
listVersions, 22

Ops.gypsum.search.clause
  (searchMetadataText), 35

POSIXct, 16, 18
prepareDirectoryUpload, 8, 23, 41, 42
publicS3Config, 8, 12, 13, 15–18, 20–22, 24, 31, 33, 34

R_user_dir, 7
refreshLatest, 12, 25
refreshUsage, 18, 26
rejectProbation, 6, 27
removeAsset, 28, 29, 30
removeProject, 11, 28, 29, 30
removeVersion, 28, 29, 30
resolveLinks, 31, 34
restUrl, 32
saveFile, 23, 33, 35
saveVersion, 8, 23, 34, 34, 42
searchMetadataText, 35
searchMetadataTextFilter
  (searchMetadataText), 35
setAccessToken (accessToken), 4
setPermissions, 16, 37
setQuota, 17, 38
startUpload, 3, 6, 8, 9, 23, 24, 27, 39, 42–44
uploadDirectory, 41
uploadFiles, 40–42, 43
validateMetadata, 15, 19, 44