Package ‘contentid’

January 21, 2025

Version 0.0.19

Title An Interface for Content-Based Identifiers

Description An interface for creating, registering, and resolving content-based identifiers for data management. Content-based identifiers rely on the 'cryptographic' hashes to refer to the files they identify, thus, anyone possessing the file can compute the identifier using a well-known standard algorithm, such as 'SHA256'. By registering a URL at which the content is accessible to a public archive (such as Hash Archive) or depositing data in a scientific repository such 'Zenodo', 'DataONE' or 'SoftwareHeritage', the content identifier can serve many functions typically associated with A Digital Object Identifier ('DOI'). Unlike location-based identifiers like 'DOIs', content-based identifiers permit the same content to be registered in many locations.

License MIT + file LICENSE

Encoding UTF-8

ByteCompile true

Depends R (>= 4.0)

Language en-US

URL `https://github.com/cboettig/contentid`

BugReports `https://github.com/cboettig/contentid/issues`

Imports openssl (>= 1.4.2), httr, curl, fs, tools, methods

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, testthat, covr, thor, vroom, spelling

VignetteBuilder knitr

NeedsCompilation no

Author Carl Boettiger [aut, cre] (<`https://orcid.org/0000-0002-1642-628X`>),
Jorrit Poelen [aut] (<`https://orcid.org/0000-0003-3138-4118`>),
NSF OAC 1839201 [fnd]
(https://www.nsf.gov/awardsearch/showAward?AWD_ID=1839201)

Maintainer Carl Boettiger <`cboettig@gmail.com`>

Repository CRAN

Date/Publication 2025-01-21 19:30:02 UTC

Contents

`content_dir` *content store home directory*

Description

A configurable default location for persistent data storage

Usage

```
content_dir(dir = Sys.getenv("CONTENTID_HOME", tools::R_user_dir("contentid")))
```

Arguments

`dir`	directory to be used as the home directory

Details

This function is intended to be called internally with no arguments. It will use the directory set by the system environmental variable CONTENTID_HOME, if set. Otherwise, it will use the default location returned by tools::R_user_dir for the application, `contentid`. Unlike rappdirs function, this function will also create the directory if it does not yet exist.

Examples

```
## example using temporary storage:
Sys.setenv(CONTENTID_HOME=tempdir())
content_dir()

## clean up
Sys.unsetenv("CONTENTID_HOME")

## Or explicitly with an argument:
content_dir(tempdir())
```

---

content_id *Generate a content uri for a local file*

Description

Generate a content uri for a local file

Usage

```
content_id(
  file,
  algos = default_algos(),
  raw = TRUE,
  as.data.frame = length(algos) > 1
)
```

Arguments

`file`	path to the file, URL, or a base::file connection
`algos`	Which algorithms should we compute contentid for? Default "sha256", see details.
`raw`	Logical, should compressed data be left as compressed binary?
`as.data.frame`	should the output be coerced into a data.frame? Default is `FALSE` if only one algorithm is computed, otherwise `TRUE`.

Details

See `https://github.com/hash-uri/hash-uri` for an overview of the content uri format and comparison to similar approaches.

Compressed file streams will have different raw (binary) and uncompressed hashes. Set `raw = FALSE` to allow base::file connection to uncompress common compression streams before calculating the hash, but this will be slower.

Value

a content identifier URI If multiple algorithms are requested, `content_id` will return a data.frame with one column per algorithm and one row for each input file. Otherwise it will return a character vector with one identifier URI for each input file. See argument `as.data.frame` above.

Examples

```
## local file
path <- system.file("extdata", "vostok.icecore.co2", package = "contentid")
content_id(path)


content_id(paste0("https://knb.ecoinformatics.org/knb/d1/mn/v2/object/",
                  "ess-dive-457358fdc81d3a5-20180726T203952542"))
```

`default_lmdb` *default location for LMDB registry*

Description

Helper utility to initialize an LMDB registry - matcher uses the pattern: "any file path ending in "lmdb". The default map size can be set using, e.g. `options(thor_mapsize=1e12)`

Usage

```
default_lmdb(dir = content_dir())
```

Arguments

`dir`	base directory for LMDB

Details

Windows machines may need to set a smaller map size, see `thor::mdb_env` for details.

`default_registries` *default registries*

Description

A helper function to conveniently load the default registries

Usage

```
default_registries()
```

Details

This function is primarily useful to restrict the scope of sources or register to, e.g. either just the remote registry or just the local registry. Note that a user can alter the registry on the fly by passing local paths and/or the URL (`https://hash-archive.org`) directly.

Examples

```
## Both defaults
default_registries()

## Only the fist one (local registry)
default_registries()[1]

## Alter the defaults with env var.
Sys.setenv(CONTENTID_REGISTRIES = tempfile())
default_registries()

Sys.unsetenv("CONTENTID_REGISTRIES")
```

`has_resource` *has_resource*

Description

Helper function to ensure examples do not execute when internet resource is temporarily unavailable, as in such cases rendering the example does not provide a reliable check. This allows examples ("tests") to "fail gracefully".

Usage

```
has_resource(url = NULL)
```

Arguments

`url`	vector of URL resources required

Examples

```
has_resource("https://google.com")
```

history_swh *return the history of archive events of a given software repository*

Description

Note that unlike the generic history method, SWH history is repo-specific rather than content-specific. An archive event adds all content from the repo to the Software Heritage archival snapshot at once. Any individual file can still be referenced by its content identifier.

Usage

```
history_swh(origin_url, host = "https://archive.softwareheritage.org", ...)
```

Arguments

origin_url	The url address to a GitHub, GitLab, or other recognized repository origin
host	the domain name for the Software Heritage API
...	additional arguments

See Also

history, store_swh, sources_swh

Examples

```
history_swh("https://github.com/CSSEGISandData/COVID-19")
```

history_url *List all content identifiers that have been seen at a given URL*

Description

history_url is the complement of sources, in that it filters a table of content identifier : url : date entries by the url.

Usage

```
history_url(url, registries = default_registries(), ...)
```

Arguments

`url`	A URL for a data file
`registries`	list of registries at which to register the URL
`...`	additional arguments

Details

`history_url()` only applies to registries that contain mutable URLs, i.e. hash-archive and local registries which merely record the contents last seen at any URL. Such URLs may have the same or different content at a later date, or may fail to resolve. In contrast, archives such as DataONE or Zenodo that resolve identifiers to source URLs control both the registry and the content storage, and thus only report URLs where content is currently found. While Download URLs from archives may move and old URLs may fail, a download URL never has "history" of different content (e.g. different versions) served from the same access URL.

Value

a data frame with all content identifiers that have been seen at a given URL. If the URL is version-stable, this should be a single identifier. Note that if multiple identifiers are listed, older content may no longer be available, though there is a chance it has been registered to a different url and can be resolved with sources.

See Also

sources

Examples

```
history_url(paste0("https://zenodo.org/api/files/5967f986-b599-4492-9a08",
"-94ce32323dc2/vostok.icecore.co2"),
registries = "https://hash-archive.carlboettiger.info")
```

`pin` *Access the latest content at a URL (DEPRECATED)*

Description

This will download the requested object to a local cache and return the local path of the object. first time it is run, and then use a local cache unless content has changed. This behavior is similar to `pins::pin()`, but uses cryptographic content hashes. Because content hashes are computed in a fast public content registry, this will usually be faster than downloading on a local connection, but slower than checking eTags in headers. Use resolve

Usage

```
pin(
  url,
  verify = TRUE,
  dir = content_dir(),
  registries = "https://hash-archive.org"
)
```

Arguments

`url`	a URL to a web resource
`verify`	logical, default TRUE. Should we verify the content identifier (SHA-256 hash) of content at the URL before we look for a local cache?
`dir`	path to the local store directory. Defaults to first local registry given to the `registries` argument.
`registries`	list of registries at which to register the URL

Details

at this time, verify mode cannot process FTP resources. Use verify = FALSE to enable a fast read from cache. This essentially allows a URL to act as an identifier, and is a good choice for URLs known to be version stable. If verify = FALSE, this will merely attempt to find a local copy of data previously associated (registered) at that URL. It will not attempt to compute the content identifier of the content at the URL, thus the local copy may or may not match the content at that address.

See Also

resolve

`purge_cache` *Purge older files from the local cache.*

Description

Deletes oldest files until cache size is below the threshold size. Additionally, users can specify a maximum age in days to delete all files older than the threshold, which can speed up file purge in large stores. Setting either age and threshold to 0 will purge everything from cache.

Usage

```
purge_cache(threshold = "1G", age = Inf, dir = content_dir(), verbose = TRUE)
```

Arguments

`threshold`	Threshold size, accepts `[fs::fs_bytes]` notation.
`age`	Maximum age in days
`dir`	the path we should use for permanent / on-disk storage of the registry. An appropriate default will be selected (also configurable using the environmental variable `CONTENTID_HOME`), if not specified.
`verbose`	show deleted file paths?

Details

Default behavior will keep `contentid`'s local store size below 1 GB. Note that `contentid` functions do not automatically call purge_cache(), this must be handled by user workflows.

Value

invisibly returns directory path

`query` *query a Content URI or a URL with remote and/or local registries*

Description

DEPRECATED, please use `sources()` or `history_url()`

Usage

```
query(uri, registries = default_registries(), ...)
```

Arguments

`uri`	a content identifier or a regular URL for a data file
`registries`	list of registries at which to register the URL
`...`	additional arguments

Value

a data frame with matching results

register *register a URL with remote and/or local registries*

Description

register a URL with remote and/or local registries

Usage

```
register(url, registries = default_registries(), ...)
```

Arguments

`url`	a URL for a data file (or list of URLs)
`registries`	list of registries at which to register the URL
`...`	additional arguments

Details

Local registries can be specified as one or more file paths where local registries should be created. Usually a given application will want to register in only one local registry. For most use cases, the default registry should be sufficient.

Value

the httr::response object for the request (invisibly)

Examples

```
register(paste0("https://knb.ecoinformatics.org/knb/d1/mn/v2/object/",
"ess-dive-457358fdc81d3a5-20180726T203952542"))
```

resolve *Resolve content from a content identifier*

Description

Requested content can be found at mutiple locations: cached to disk, or available at one or more URLs. This function provides a mechanism to always return a single, local path to the content requested, (provided the content identifier can be found in at least one of the registries).

Usage

```
resolve(
  id,
  registries = default_registries(),
  verify = TRUE,
  store = FALSE,
  dir = content_dir(),
  ...
)
```

Arguments

`id`	A content identifier, see content_id
`registries`	list of registries at which to register the URL
`verify`	logical, default TRUE. Should we verify that content matches the requested hash?
`store`	logical, should we add remotely downloaded copy to the local store?
`dir`	path to the local store directory. Defaults to first local registry given to the `registries` argument.
`...`	additional arguments

Details

Local storage is checked first as it will allow us to bypass downloading content when a local copy is available. If no local copy is found but one or more remote URLs are registered for the hash, downloads from these will be attempted in order from most recent first.

See Also

query query_local query_remote

Examples

```
# ensure some content in local storage for testing purposes:
vostok_co2 <- system.file("extdata", "vostok.icecore.co2",
                          package = "contentid")
store(vostok_co2)


resolve(paste0(
 "hash://sha256/9412325831dab22aeebdd6",
 "74b6eb53ba6b7bdd04bb99a4dbb21ddff646287e37")
)
```

`retrieve` *Retrieve files from the local cache*

Description

Retrieve files from the local cache

Usage

```
retrieve(id, dir = content_dir())
```

Arguments

`id`	a content_id
`dir`	the path we should use for permanent / on-disk storage of the registry. An appropriate default will be selected (also configurable using the environmental variable `CONTENTID_HOME`), if not specified.

Value

path to a local copy of the file.

See Also

store

Examples

```
# Store & retrieve local file
vostok_co2 <- system.file("extdata", "vostok.icecore.co2",
                          package = "contentid")
 id <- store(vostok_co2)
 retrieve(id)
```

retrieve_swh *retrieve content from Software Heritage given a content identifier*

Description

retrieve content from Software Heritage given a content identifier

Usage

```
retrieve_swh(id, host = "https://archive.softwareheritage.org")
```

Arguments

id	a content identifier
host	the domain name for the Software Heritage API

See Also

retrieve, sources_swh

Examples

```
id <- paste0("hash://sha256/9412325831dab22aeebdd",
             "674b6eb53ba6b7bdd04bb99a4dbb21ddff646287e37")
retrieve_swh(id)
```

sources *List all known URL sources for a given Content URI*

Description

List all known URL sources for a given Content URI

Usage

```
sources(
  id,
  registries = default_registries(),
  cols = c("source", "date"),
  all = TRUE,
  ...
)
```

Arguments

`id`	a content identifier
`registries`	list of registries at which to register the URL
`cols`	names of columns to keep. Default are `source` and `date`. See details.
`all`	should we query remote registries even if a local source is found? Default TRUE
`...`	additional arguments

Details

possible columns are (in order): `identifier`, `source`, `date`, `size`, `status`, `md5`, `sha1`, `sha256`, `sha384`, `sha512`

Value

a data frame with all registration events when a URL or a local path (including the local store) have contained the corresponding content.

See Also

history register store

Examples

```
id <- paste0("hash://sha256/9412325831dab22aeebdd",
            "674b6eb53ba6b7bdd04bb99a4dbb21ddff646287e37")
```

```
sources(id)
```

---

sources_swh *List software heritage sources for a content identifier*

Description

List software heritage sources for a content identifier

Usage

```
sources_swh(id, host = "https://archive.softwareheritage.org", ...)
```

Arguments

`id`	a content identifier
`host`	the domain name for the Software Heritage API
`...`	additional arguments

See Also

sources

Examples

```
id <- paste0("hash://sha256/9412325831dab22aeebdd",
             "674b6eb53ba6b7bdd04bb99a4dbb21ddff646287e37")
sources_swh(id)
```

store *Store files in a local cache*

Description

Resources at a specified URL will be downloaded and copied into the local content-based storage. Local paths will simply be copied into local storage. Identical content is not duplicated.

Usage

```
store(x, dir = content_dir(), algos = default_algos())
```

Arguments

x	a URL, connection, or file path.
dir	the path we should use for permanent / on-disk storage of the registry. An appropriate default will be selected (also configurable using the environmental variable CONTENTID_HOME), if not specified.
algos	Which algorithms should we compute contentid for? Default "sha256", see details.

Value

the content-based identifier

See Also

retrieve

Examples

```
# Store & retrieve local file
vostok_co2 <- system.file("extdata", "vostok.icecore.co2",
                          package = "contentid")
 id <- store(vostok_co2)
 retrieve(id)
```

`store_swh` *Add content to the Software Heritage Archival Store*

Description

Add content to the Software Heritage Archival Store

Usage

```
store_swh(
  origin_url,
  host = "https://archive.softwareheritage.org",
  type = "git",
  ...
)
```

Arguments

`origin_url`	The url address to a GitHub, GitLab, or other recognized repository origin
`host`	the domain name for the Software Heritage API
`type`	software repository type, i.e. "git", "svn"
`...`	additional arguments

Examples

```
store_swh("https://github.com/CSSEGISandData/COVID-19")
```

`swh_ratelimit` *software heritage rate limit*

Description

software heritage rate limit

Usage

```
swh_ratelimit(verbose = TRUE)
```

Arguments

`verbose`	show messages about current rate limits? Software Heritage has a rate limit that can interfere with common queries (resolve sources, register) when large numbers of queries are made. contentid functions will not automatically check against wait limits, but may fall back to other registeries when available.

Value

remaining, total, and time until next reset are invisibly returned as a data.frame.

Index