Package ‘pcloudr’

May 14, 2023

Title Accessing 'pCloud.com' from R with 'OAuth2'

Version 0.0.1

Description Programmatic delegated access to 'pCloud.com', based on 'OAuth2'.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports curl,
httpuv,
jsonlite,
secretR,
sodium

Remotes local::../secretR

Suggests knitr,
rmarkdown,
spelling,
testthat (>= 3.0.0)

Language en-GB

VignetteBuilder knitr

Config/testthat/edition 3

R topics documented:

pcloudr-package 2
authorising 2
copy 4
delete 5
encoding 6
endpoint 8
folders 8
formats 10
pcloud-api 11
pcloud.config 13
pcloud.contest 13
pcloud.resp 13
pcloud.try 14

public-downloads 14
public-uploads 16
read-write 18
remote-paths 19
tests 20
time 21

Index **22**

`pcloudr-package` *Accessing pCloud from R with OAuth2*

Description

pcloudr lets you access to pCloud cloud programmatically from R through OAuth2 delegated access.

Details

To use pcloudr you first have to register an app in pCloud or anyway obtain app credentials. Note that app credentials are not intended and cannot be secret for non-web apps. See the section "Security considerations" in the vignette.

API functions try to catch JSON errors when they occur and stop, otherwise they return the intended result, e.g. the content of a file or the success flag, "0" as character.

If you want to get more information about the pCloud API methods wrapped by pcloudr, refer to `https://docs.pcloud.com/methods/`

Note that pcloudr functions are not a 1-to-1 mapping from pCloud API, but they try to adopt a higher-level approach, possibly streamlining some cumbersome tasks. However, if you want to replicate an official method described in the link above use `pcloud.lapi()` or the simplified version `pcloud.api()`. You just need to pass the method and the documented key-value fields, while delegating the subtleties of OAuth2 authentication to pcloudr.

Author(s)

Maintainer: Antonio Fasano `<first.last@sample.com>`

`authorising` *Authorise a registered app*

Description

Interactive procedure to authorise a client app registered on pCloud.

Usage

```
pcloud.auth(
  client.id,
  client.secret,
  ports,
  prompt = NULL,
  policy = secretR::pwpolicy(),
  mask = TRUE,
  no.file = FALSE
)
```

Arguments

client.id	the app client identification issued at registration.
client.secret	the app client secret issued at registration.
ports	a numeric vector of the redirect ports set during the app registration. A port is the xxxx component in the http://localhost:xxxx redirect URIs.
prompt	if non-null, it the the user prompt asking the password to encrypt pCloud access codes.
policy	the security policy used to set the encryption password for pCloud access codes. Use secretR::pwpolicy()$pwdesc to print the current policy and ?secretR::pwpolicy to learn how to change it.
mask	masking the characters from prying eyes when inputting the password?
no.file	If TRUE, save pCloud access codes to ~/pcloudr or the values set with pcloud.config().

Details

It is suggested that you have a look at the 'pcloudr' vignette before using this function. Start it with browseVignettes("pcloudr").

A pCloud app can be registered at https://docs.pcloud.com/my_apps/, possibly setting 'Folder access' to 'Private' (and later shown as 'Access' to 'Specific folder only').

The client.id and client.secret are visible in the app page, and in the app 'Settings' one or more redirect URIs should be set in the form http://localhost:xxxx, where xxxx is the desired TCP port number to be used in the ports vector. For example: http://localhost:65432 and http://localhost:65433. Also, for better security, disallow the 'implicit grant' setting.

If no.file = TRUE, at the end of the session the authorisation is lost and you have to repeat the procedure, also prompt, policy, and mask args are ignored without errors.

If no.file = FALSE, the default, pCloud access codes (access token and endpoint) are saved encrypted to ~/pcloudr, but you can set an alternative path with pcloud.config(). In this case, you are prompted (with a graphical interface) to provide an encryption password. The prompt describes the password security policy, which you can also obtain from secretR::pwpolicy()$pwdesc. You might also set your own policy with secretR::pwpolicy() and supply it to the function's policy argument.

Value

logical value for success.

See Also

[secretR::pwpolicy()],pcloud.config(),browseVignettes("pcloudr")

---

copy *Copy and Rename Items in Cloud*

Description

pcloud.copyfile and pcloud.copyfolder do what they mean. pcloud.renamefile and pcloud.renamedir work like Linux mv command, they can be used to rename and move a filesystem object.

Usage

```
pcloud.copyfile(
  remote.path = NULL,
  fileid = NULL,
  topath = NULL,
  tofolderid = NULL,
  noover = FALSE
)

pcloud.renamefile(
  remote.path = NULL,
  fileid = NULL,
  topath = NULL,
  tofolderid = NULL,
  toname = NULL
)

pcloud.copydir(
  remote.path = NULL,
  folderid = NULL,
  topath = NULL,
  tofolderid = NULL,
  noover = FALSE,
  skipexisting = FALSE,
  copycontentonly = FALSE
)

pcloud.renamedir(
  remote.path = NULL,
  folderid = NULL,
  topath = NULL,
  tofolderid = NULL,
  toname = NULL
)
```

Arguments

remote.path	the pCloud absolute remote path
fileid	the pCloud file id (an alternative to path obtained from listfolder method).
topath	destination path of the copied/renamed object, with a trailing slash if a folder.
tofolderid	ID of the folder to which the the object is moved.
noover	if TRUE, overwrite existing file or folder.
toname	destination name of the renamed object.
folderid	the pCloud folder id (an alternative to path obtained from listfolder method).
skipexisting	if TRUE, skip files that already exist.
copycontentonly	if TRUE, copy only the source folder's content, rather than the folder itself.

Details

For pcloud.renamefile, the renaming target can be one of topath, tofolderid, or tofolderid/toname. If it is tofolderid/toname or topath is a file, than the file is renamed as the implied file. If it is tofolderid or topath is a folder, the file is moved to the implied folder, but, in this case, topath MUST end with a trailing slash.

For pcloud.renamedir, the renaming target can be one of topath, tofolderid, or tofolderid/toname. If it is tofolderid/toname or topath is a non-existing folder, than the folder is renamed as the implied folder. If it is tofolderid or topath is a folder, the folder is moved inside the implied folder, but, in this case, topath MUST end with a trailing slash.

Value

JSON response content invisibly.

---

delete *Delete Items in Cloud*

Description

pcloud.delete.file, pcloud.delete.folder, and pcloud.delete.folderrec functions delete files, folders, and folders recursively. pcloud.delete.folderfiles() delete all files in a folder. pcloud.delete.foldercont() is similar to pcloud.delete.folderrec(), but preserve the container while deleting the content.

Usage

```
pcloud.delete.file(remote.path = NULL, fileid = NULL)

pcloud.delete.folder(remote.path = NULL, folderid = NULL)

pcloud.delete.folderrec(remote.path = NULL, folderid = NULL)

pcloud.delete.folderfiles(remote.path = NULL, folderid = NULL)

pcloud.delete.foldercont(remote.path = NULL, folderid = NULL)
```

Arguments

remote.path	the pCloud absolute remote path
fileid	the pCloud file id (an alternative to path obtained from pcloud.listfolder())..
folderid	the pCloud folder id (an alternative to path obtained from pcloud.listfolder())..

Value

pcloud.delete.file: TRUE if deletion was successful, FALSE otherwise.

pcloud.delete.folder: TRUE if deletion was successful, FALSE otherwise.

pcloud.delete.folderrec: a vector with the names delfiles and delfolds for the number of deleted files and folders.

pcloud.delete.folderfiles: the number of successfully deleted files, with an error if less than available files.

pcloud.delete.foldercont: a vector with the names delfiles and delfolds for the number of deleted files and folders.

---

encoding *Encoding Links ‘*

Description

Generate encoded links for safer copy and paste.

Usage

```
pcloud.encodelink(code)

pcloud.decodelink(enclink, error = TRUE)

pcloud.url.code(url, autocode = FALSE)

pcloud.direct.dec(enclink, rawoutput = FALSE)

pcloud.read.link.dec(enclink, text = FALSE)

pcloud.down.from.plink.dec(enclink, local.path)

pcloud.pupload.data.dec(enclink, username, data, filename, rawoutput = FALSE)

pcloud.pupload.file.dec(
  enclink,
  username,
  local.path,
  filename,
  curlwarn = TRUE,
  rawoutput = FALSE
)

pcloud.pupload.info.dec(enclink)
```

Arguments

code	the sharing code perhaps returned pcloud.pub.down.filelink(). See details.
enclink	the sharing code returned by pcloud.pub.down.filelink().
error	if FALSE return NA on decoding error, rather than stopping.
url	the URL from which we want to extract the code field.
autocode	if TRUE, and url does not contain any of the chars in /?=, url is not parsed and considered a code as is.
rawoutput	also show raw API output.
text	return the code as text.
local.path	local path of the file to download or upload (send).
username	name of the uploader.
data	data to write as character or raw.
filename	name of the destination file.
curlwarn	warn if the 'cURL' library does not support 'filename' args.

Details

When you publish a public upload or download link, intended for subsequent copy-and-paste, the user might easily misselect it. The encoding adds some redundant characters and a checksum to try to fix the selection or inform the user of the copy-error.

The encoded link scheme is:

```
===code@host@check===
```

where 'code' is the code from pcloud.pub.down.filelink(); 'host' refers only to the second level API domain, assuming the domain name is always *.pcloud.com; check is a four-digit checksum; ='s are dummy redundant characters which reduce accidental misselection.

If you are publicly sharing a file with pcloud.pub.down.filelink(), code is one of the named components returned by this function. The code is also contained as a parameter in the upload and download URLs, intended for browser use. You can use pcloud.url.code() to extract the code value from a public link. Because currently pCloud does not support creating upload links with OAuth2, this comes handy to parse links obtained from alternative authentication methods or pCloud apps.

pcloud.direct.dec(), pcloud.read.link.dec(), pcloud.down.from.plink.dec(), pcloud.pupload.info.dec(), pcloud.pupload.data.dec(), and pcloud.pupload.file.dec(), are equivalent to their counterpart without the .dec suffix, except they replace the encoded string from pcloud.encodelink() for the sharing-code/endpoint pair.

Value

pcloud.encodelink: the encoded link string.

pcloud.decodelink: a named list where 'code' is the sharing code returned by pcloud.pub.down.filelink and 'endpoint' the API endpoint of the sharing user.

pcloud.url.code: the extracted code string.

pcloud.direct.dec: the URL to download the file, and a message with more link information if rawoutput = TRUE.

`pcloud.read.link.dec`: the read data in raw format, or character if `text` is `TRUE`.

`pcloud.down.from.plink.dec`: the return value of `curl::curl_fetch_disk` invisibly.

`pcloud.pupload.data.dec`: if `rawoutput=FALSE`, only the logical success; else prints JSON response content, success feedback, and invisibly return logical success.

`pcloud.pupload.file`: if `rawoutput=FALSE` only the logical success; else prints JSON response content, success feedback, and invisibly return logical success.

`pcloud.pupload.info.dec`: a list with link info.

`endpoint` *Endpoint*

Description

`pcloud.endpoint()` returns the endpoint associated with the OAuth account.

Usage

```
pcloud.endpoint()
```

Details

pCloud provides different data regions, where files and data are stored, which the user is able to select when signing on or later for a fee. The endpoint denotes the server to which the API requests are sent and, for pCloud, it depends on the data region.

The endpoint is important for shared resources, in fact, to obtain direct links, which do not require to interactively access pCloud website, it is is necessary both the sharing code and the endpoint.

Value

endpoint string.

`folders` *Manage Folders*

Description

Functions to list and create folders.

Usage

```
pcloud.root(narrow = FALSE)

pcloud.listfolder(
  remote.path = NULL,
  folderid = NULL,
  recursive = NULL,
  showdeleted = NULL,
  nofiles = NULL,
  noshares = NULL
)

pcloud.narrow(
  remote.path = NULL,
  folderid = NULL,
  recursive = NULL,
  showdeleted = NULL,
  nofiles = NULL,
  noshares = NULL
)

pcloud.listfolderfiles(remote.path = NULL, folderid = NULL, narrow = FALSE)

pcloud.listfolderfileids.rec(
  remote.path = NULL,
  folderid = NULL,
  namedvec = TRUE,
  suffix = FALSE
)

pcloud.createfolderifnotexists(
  remote.path = NULL,
  folderid = NULL,
  name = NULL
)
```

Arguments

narrow	if TRUE only show files' "path", "modified", "size" and folder attribute. See details.
remote.path	the pCloud remote folder path.
folderid	the pCloud folder ID.
recursive	If is set full directory tree will be returned, which means that all directories will have contents filed.
showdeleted	if is set, deleted files and folders that can be undeleted will be displayed.
nofiles	if is set, only the folder (sub)structure will be returned.
noshares	if is set, only user's own folders and files will be displayed.
namedvec	return a named vector file IDs and filenames
suffix	return filenames with file IDs and dash suffix.
name	name of the folder to create.

Details

Folder listing functions, such as `pcloud.listfolder`, return metadata which in turn have a content field which is an array of the metadata of folder's contents.

`pcloud.narrow()` and `pcloud.root(narrow = TRUE)`, `pcloud.files(narrow = TRUE)` give a data frame with paths, modification dates as POSIXlt classes, sizes, and, except for `pcloud.files`, the logical `is.fld` column `TRUE` for folders.

`pcloud.createfolderifnotexists` creates a folder if the folder doesn't exist or returns the existing folder's metadata.

Value

`pcloud.root`: the same value as `pcloud.root("/")`.

`pcloud.listfolder`: a data.frame representing the folder. If `recursive = TRUE`, the `contents` element links to subfolders.

`pcloud.narrow`: a narrow version of `pcloud.listfolder()`, including only the paths, modification time, size, and a logical folder attribute. See details for column formats.

`pcloud.listfolderfiles`: if `narrow` is `FALSE`, the same value of `pcloud.listfolder()`, but excluding folders; if it is `TRUE`, a narrower version with the same columns as `pcloud.narrow()`, except `is.flds`.

`pcloud.listfolderfilesids.rec`: if `namedvec` is `TRUE`, a character vector whose names are the file IDs and values the file names; otherwise return only the file IDs; if `suffix` is `TRUE`, a vector of filenames suffixed with file IDs and a dash (e.g. `fileid-filename`), and with file IDs as names if `namedvec` is `TRUE`.

`pcloud.createfolderifnotexists`: metadata of the newly created or existing folder.

`formats` *Formats and Style*

Description

Internally, pcloudr uses two formats: the *full response*, which is a list with the server response content and other relevant response parameters; the *response JSON list*, which is a list version of the JSON response content, when available.

Function return values are classified into two categories, functional and non-functional, only the former is interesting, the latter returns invisibly the response content (perhaps as a *response JSON list), unless exceptions are raised,

To get more more info on responses, you may use `pcloud.resp()`, and you may manage exceptions in conditions with `pcloud.try()`.

Details

Full response

Internally, pcloudr formats the server response as a list with following three fields (elements):

- `ok`: `TRUE` if the returned status code is less then 400;
- `cnt`: response content;
- `code` the server status code ;

- `type` response type.

For convenience , we define this list the *full response* because it contains most of the things you may need and usually more, but full headers are missing.

Unless `type` is "application/octet-stream", `cnt` is converted to character (this is going to be more refined in the future to avoid edge cases).

If `type` is `application/json`, `cnt` string is converted into a *response JSON list*, which is an R list version of the JSON structure more easily actionable. (See https://arxiv.org/abs/1403.2805). Of course, we this is for simplicity, as the list represents only the response content.

Response JSON list

When the field `cnt` of the full response is a JSON list the actual elements are set by the API server, but two elements appear to be standard: `result` and `error`. If the API server reports no error, the former is zero and the latter is missing.

When pcloudr spots a non-zero `result`, it throws an exception based on `error`, which in R usually translates into a `stop()` command.

As noted, not always the response content is JSON and hence actionable as a list. This is the case, for example, when you mistype the API endpoint (and you get a 404 error), or the destination is unreachable. Also. when you are downloading a binary file, then the API server gives a JSON response only if there is an error.

Style

pcloudr functions can be divided in two categories: those with *non-functional return*, which are relevant for their side effect, e.g. uploading a file; those with *functional return*, which are without side effects, such as listing the content of a folder or querying for the existence of filesystem object.

If no exception are raised: functional return consists of the relevant queried content, such as the content of the file, the logical result of an existence test, etc.; non-functional return is uninteresting, so the response JSON list is returned.

Helpers

Normally you are fine with the `cnt` field of the full response, however, you may use `pcloud.resp()` to get the latter. Most pcloudr functions are high level in that they involve multiple request to the API server. `pcloud.resp()` can return a list of all intermediate requests with the relevant URL parameters.

It is common for web functions to get errors that need to be managed in loops or conditionals construct. For example you might want to try a request more times if it fails. On those cases `pcloud.try()`, rather than stopping the execution flow, prints the stop message, for the user, and return `FALSE` for the program to act.

`pcloud-api` *Generic API call*

Description

Call any method documented at https://docs.pcloud.com/methods/.

Usage

```
pcloud.api(method, ..., mask = TRUE, logical = TRUE, rawoutput = FALSE)

pcloud.lapi(
  method,
  pars,
  mask = TRUE,
  slow = FALSE,
  curl.opt = NULL,
  curl.form = NULL,
  logical = TRUE,
  nosimplejson = FALSE,
  rawoutput = FALSE
)
```

Arguments

`method`	the API method to call
`...,`	named R arguments converted into method parameters.
`mask`	masking the characters from prying eyes when inputting the password to read pCloud access codes from disk.
`logical`	return logical success.
`rawoutput`	verbose output as from the style guide.
`pars`	named list of method parameters.
`slow`	add some connection tests before requests.
`curl.opt`	option list for curl::handle_setopt
`curl.form`	option list for curl::handle_setform
`nosimplejson`	do not apply simplification to JSON output.

Details

These functions allow to call any pCloud API method (unless it does not support OAuth2). Use them if you want to use documented methods as-is, but get rid of the intricacies of OAuth2.

`pcloud.api()` is a more streamlined version of `pcloud.lapi()` with less features. The convenience of `pcloud.api()` is that you don't have to wrap methods arguments inside a list. However, bear in mind that, should a method have the same arguments as `pcloud.api()`, for example `mask`, then the function would "steal" from the method, taking precedence, thus you have to switch to `pcloud.lapi()` to pass the argument.

If the response content-type is JSON, the value is converted to a named list with `jsonlite::fromJSON()`.

Value

The requests response possibly jsonified.

pcloud.config *Package configuration*

Description

Set package options. Currently the only option is the custom path to pCloud access codes, an encrypted file. The option is stored in memory, thus it needs to be re-set on each session.

Usage

```
pcloud.config(accodes.path)
```

Arguments

`accodes.path` the path to the encrypted file containing the pCloud access codes.

pcloud.contest *Test Connection*

Description

Connects to the root folder using some connectivity tests and print statistics. This makes the command slow, but potentially useful to detect problems and bottleneck.

Usage

```
pcloud.contest()
```

Value

Print the results of connection tests with elapsed time and return invisibly `pcloud.root()` output.

pcloud.resp *Debugging server responses*

Description

While internally pcloudr functions retrieve the server response, this is usually not shown to the user. By wrapping the call inside `pcloud.resp(...)`, it is possible to obtain the original server response.

Usage

```
pcloud.resp(expr, cont = FALSE, multi = FALSE)
```

Arguments

`expr`	the function call for which the server response is required
`cont`	if TRUE, return only the jsonified element of the full response list (see Formats and Style.)
`multi`	when multiple server requests occur, return only the last one (FALSE) or all of them (TRUE).

Details

For more information regarding internal pcloudr format see Formats and Style.

Value

If `multi = TRUE` a list with all the responses given to each request. Responses consist of the jsonified server response if `cont = TRUE` or only its `cnt` element if `FALSE`. Responses have also a query element with the relevant part of the URL requested.

`pcloud.try` *Managing exceptions*

Description

If the expression `expr` throws an exception, `pcloud.try(expr)` prints its stop message and continue returning FALSE. This is useful when in loops and conditionals, to take action upon failure such as repeating the command a n times.

Usage

```
pcloud.try(expr)
```

Arguments

`expr`	expression potentially throwing an exception.

`public-downloads` *Public Download Links*

Description

Generate and download from public links.

Usage

```
pcloud.pub.down.filelink(
  remote.path = NULL,
  fileid = NULL,
  expire = NULL,
  maxdownloads = NULL,
  maxtraffic = NULL,
  shortlink = NULL,
  linkpassword = NULL
)

pcloud.direct(code, endpoint, rawoutput = FALSE)

pcloud.read.link(code, endpoint, text = FALSE)

pcloud.down.from.plink(code, endpoint, local.path)

pcloud.delink(linkid)
```

Arguments

`remote.path`	the pCloud remote file path.
`fileid`	the pCloud file id (an alternative to path obtained from listfolder method).
`expire`	time when the link will stop working as POSIXt class.
`maxdownloads`	maximum number of downloads for this file.
`maxtraffic`	maximum traffic in bytes that this link will consume.
`shortlink`	if set, a short link will also be generated
`linkpassword`	set a password for the link.
`code`	the sharing code returned by `pcloud.pub.down.filelink()`.
`endpoint`	the sharing user endpoint, who can obtain it from `pcloud.endpoint()`.
`rawoutput`	also show raw API output.
`text`	return the code as text.
`local.path`	local path of the file to download.
`linkid`	the ID of the link to be deleted, returned by `pcloud.pub.down.filelink()`.

Details

`pcloud.pub.down.filelink()` generates a public link to download a pCloud file, intended for non-programmatic access, via the pCloud web site.

`pcloud.direct()` generates a temporary direct link to download a publicly shared item, intended for programmatic access. To generate a direct download link to a cloud item, it is necessary the item's sharing code and the endpoint associated with the account of the sharing user. The sharing code is in the output list of `pcloud.pub.down.filelink()` (it is also included in the website link generated by `pcloud.pub.down.filelink()`). The endpoint can be obtained from `pcloud.endpoint()`. The duration of direct links is not specified by pCloud, but a user should generate them only when they intend "to actually download the file". Luckily, provided that you have delivered both the code and the endpoint, anyone can obtain a direct link, when they are about to download, without authentication.

Typically, you are not actually interested in the (direct) file link, but rather in its content. Thus, what you want are `pcloud.read.link()` and `pcloud.down.from.plink()` respectively to read data from a public link and to download it to a file. When you do not need the link any more, `pcloud.delink()` can delete it.

`pcloud.direct()`, `pcloud.read.link()` and `pcloud.down.from.plink()` do not require authentication.

A convenient way to distribute sharing codes+endpoints could be to generate an encoded string including them. See `pcloud.encodelink()`

Value

`pcloud.pub.down.filelink`: a list of three elements: `website`, the website link for non-programmatic download; `linkid`, the numeric ID that can be used to delete/modify this public link; `code`, the sharing code to generate a direct download link; `cnt`, The entire API JSON output as a list.

`pcloud.direct`: the URL to download the file, and a message with more link information if `rawoutput = TRUE`.

`pcloud.read.link`: read data in raw format, or character if `text` is `TRUE`.

`pcloud.down.from.plink`: the return value of `curl::curl_fetch_disk` invisibly.

`pcloud.delink`: JSON response content invisibly.

---

public-uploads *Public Upload Links*

Description

Upload data, with `pcloud.pupload.data()`, and files, with `pcloud.pupload.file()`, to public links. Get link information with `pcloud.pupload.info()`.

Usage

```
pcloud.pupload.data(
  code,
  endpoint,
  username,
  data,
  filename,
  rawoutput = FALSE
)

pcloud.pupload.file(
  code,
  endpoint,
  username,
  local.path,
  filename,
  curlwarn = TRUE,
  rawoutput = FALSE
)

pcloud.pupload.info(code, endpoint)
```

Arguments

code	the value of the 'code' field in the public upload link.
endpoint	the endpoint of the destination user, who can obtain it from pcloud.endpoint().
username	name of the uploader.
data	data to write as character or raw.
filename	name of the destination file.
rawoutput	also show raw API output.
local.path	local path of the file to send.
curlwarn	warn if the 'cURL' library does not support 'filename' args.

Details

pCloud allows to create a public link to a folder to request file. The link is similar to an email, as a third party can only upload but can't see the content.

Unfortunately, the link creation is not supported through OAuth2, thus one has to use an alternative method. For example, with the webapp, open the folder and select 'Request files' under the ellipsis folder menu.
pcloud.pupload.* functions share some similarities with pcloud.write and pcloud.upload, but pcloud.write implies opening and closing file descriptors. Also, pcloud.pupload.* functions do not require authentication.

The philosophy of this package is to not ask the user their account credentials, therefore we have no precooked function to create upload links. However, if you need programmatic creation of upload links and you are fine with password digest authentication, see the Examples section below to create an upload link to a folder using its folderid.

A convenient way to distribute sharing codes+endpoints could be to generate an encoded string including them. See pcloud.encodelink()

NOTE. For pcloud.pupload.data(), the data argument can only be of type raw or character. If it is NULL, zero-length, or an empty string, it is replaced with raw(1), which is equivalent a file with a single null-byte.

Value

pcloud.pupload.data: if rawoutput=FALSE, only the logical success; else prints JSON response content, success feedback, and invisibly return logical success.

pcloud.pupload.file: if rawoutput=FALSE only the logical success; else prints JSON response content, success feedback, and invisibly return logical success.

pcloud.pupload.info: a list with link info.

Examples

```
## Not run:
## Link creation without OAuth2
username <- "***"; pass <-"***"
endpoint <- "***" # you can use pcloud.endpoint() to obtain it
folderid <- "***" # you can use pcloud.listfolder("/") to obtain it
comment <- "Upload files" # required

## curl and digest packages required
curl <- function(url) rawToChar(curl::curl_fetch_memory(url)$content)
```

```
sha1 <- function(msg) digest::digest(msg, algo = "sha1", serialize = FALSE)

## Create digest credentials
digest <- curl(sprintf("%s/getdigest", endpoint))
digest <- strsplit(digest, '"')[[1]][10]
passworddigest <- sha1(paste0(pass, sha1(tolower(username)), digest))
cred <- sprintf("logout=1&username=%s&digest=%s&passworddigest=%s",
                username, digest, passworddigest)

## Create upload link
curl(sprintf("%s/createuploadlink?%s&comment=%s&folderid=%s",
             endpoint, cred, curl::curl_escape(comment), folderid))

## End(Not run)
```

`read-write` *Read & Write Data*

Description

`pcloud.read()` and `pcloud.write()` read from and write to a remote pCloud file, and `pcloud.fsize` gets the size. These high-level functions take care of opening and closing the file. `pcloud.read.folderfiles()` and `pcloud.download.folderfiles()` read and download all file in a folder. `pcloud.upload()` uploads a local file. These functions are not intended to download/upload large files.

Usage

```
pcloud.fsize(remote.path = NULL, fileid = NULL)

pcloud.read(
  remote.path = NULL,
  fileid = NULL,
  count = NULL,
  verify = FALSE,
  text = NA
)

pcloud.read.folderfiles(remote.path = NULL, folderid = NULL, verbose = FALSE)

pcloud.download.folderfiles(
  remote.path = NULL,
  folderid = NULL,
  local.dir,
  text = FALSE,
  verbose = TRUE
)

pcloud.write(data, remote.path = NULL, fileid = NULL, verbose = FALSE)

pcloud.upload(
  local.path,
```

```
  remote.path = NULL,
  folderid = NULL,
  filename = NULL,
  renameifexists = FALSE,
  curlwarn = TRUE
)
```

Arguments

remote.path	the pCloud remote folder path.
fileid	the pCloud file id (an alternative to path obtained from listfolder method).
count	read only count bytes.
verify	verify the download with sha256 checksum
text	if TRUE save as text, else save as a binary file.
folderid	the pCloud folder id (an alternative to path obtained from listfolder method).
verbose	show length of uploaded data.
local.dir	path of the folder to download files.
data	data to write as character or raw
local.path	local path of the file to send.
filename	filename to give to the uploaded file.
renameifexists	if the destination file exists, uploaded file will be renamed.
curlwarn	Warn if cURL does not support filename args.

Value

pcloud.fsize(): the size in bytes of the remote file as numeric.

pcloud.read: the content of the remote file as 'application/octet-stream'.

pcloud.read.folderfiles: a list whose element are named after the read files and values are the content of files.

pcloud.download.folderfiles: NULL invisibly.

pcloud.write: logical success and if verbose, prints the byte written.

pcloud.upload: logical success.

---

remote-paths *Remote Paths*

Description

When your app uses private folder access, the remote root path (/) is relative to the pCloud-assigned app folder. If you use folder IDs, the root folder is always identified by 0.

Details

When you register your app (see "Registering your app" in 'pcloudr' vignette or `pcloud.auth()`), you can to restrict its access to an app specific folder. At the time of the registration, this setting is named *private folder access*, and is later shown as 'Access' to 'Specific folder only' in the app settings available at https://docs.pcloud.com/my_apps/.

If compatible with your workflow, that is your app does not read access the entire drive, this is a convenient security setting, which, by the way, affects the way you specify remote paths in pcloudr.

In general, you specify paths with the usual R (Unix) style, such as `/dir/subdir/hello.txt`, where the first forward slash (`/`) denotes the root of your cloud drive, assuming your app has access to all folders. However, when you are using private folder access, your app operations are restricted to the cloud folder `/Applications/<appname>`, where `appname` is the name you chose for your app. Given this, the file `/Applications/<appname>/hello.txt` is now seen by your app as `/hello.txt`, put it another way, the root folder for the restricted app is `/Applications/<appname>/`.

Each file and folder have also a filesystem ID. As a rule, whenever you are required to give a path, pcloudr functions allows you to give also the ID of the related filesystem object. The ID is a unique number which is not affected by the private folder access setting.

The IDs are provided by the list-folder functions. Typically, these functions provide for each cloud item a `folderid`, `fileid`, and `id`. The first two are integers, available only when the filesystem item is of the specified type, and `NA` otherwise, e.g. the `folderid` of a file is `NA`. When pcloudr function arguments ask cloud IDs, you have to pass `folderid` or `fileid` depending on the type of item required. For your convenience, an `id` field is also provided, which gives a unified view, where the IDs of folders are prefixed with a 'd'.

There is a special ID, `0`, which always identifies the root folder, and therefore it is affected by restrictions to a private folder.

---

tests *Test Items in Cloud*

Description

`pcloud.exists.path`, `pcloud.exists.file`, `pcloud.exists.folder` test for existence of a path a file and a folder respectively.

Usage

```
pcloud.exists.path(remote.path)

pcloud.exists.folder(remote.path)

pcloud.exists.file(remote.path)

pcloud.count.items(remote.path = NULL, folderid = NULL)

pcloud.folder.empty(remote.path = NULL, folderid = NULL)

pcloud.count.files(remote.path = NULL, folderid = NULL)

pcloud.count.folders(remote.path = NULL, folderid = NULL)
```

Arguments

remote.path	the pCloud absolute remote path
folderid	the pCloud folder id (an alternative to path obtained from pcloud.listfolder()).

Value

Logical success or item count as numeric

---

time *Time Conversion*

Description

Convert to and from pCloud time. These functions come handy when you want to make your own API calls with pcloud.*api().

Usage

```
pcloud.to.pctime(posix)

pcloud.from.pctime(pctime)
```

Arguments

posix	R POSIXt time
pctime	pCloud time string, such as "Sun, 01 Jan 2023 14:50:44 +0000", to POSIXct

Value

pcloud.to.pctime: a string similar to "2023-02-13T18:23:37+0100"

pcloud.from.pctime: a POSIXct time class.

Index

authorising, 2

copy, 4

delete, 5

encoding, 6
endpoint, 8

folders, 8
formats, 10
Formats and Style, *14*

list-folder functions, *20*

pcloud-api, 11
pcloud.api *(pcloud-api)*, 11
pcloud.api(), *2*
pcloud.auth *(authorising)*, 2
pcloud.auth(), *20*
pcloud.config, 13
pcloud.config(), *3*, *4*
pcloud.contest, 13
pcloud.copydir *(copy)*, 4
pcloud.copyfile *(copy)*, 4
pcloud.count.files *(tests)*, 20
pcloud.count.folders *(tests)*, 20
pcloud.count.items *(tests)*, 20
pcloud.createfolderifnotexists *(folders)*, 8
pcloud.decodelink *(encoding)*, 6
pcloud.delete.file *(delete)*, 5
pcloud.delete.folder *(delete)*, 5
pcloud.delete.foldercont *(delete)*, 5
pcloud.delete.folderfiles *(delete)*, 5
pcloud.delete.folderrec *(delete)*, 5
pcloud.delink *(public-downloads)*, 14
pcloud.direct *(public-downloads)*, 14
pcloud.direct.dec *(encoding)*, 6
pcloud.down.from.plink *(public-downloads)*, 14
pcloud.down.from.plink.dec *(encoding)*, 6
pcloud.download.folderfiles *(read-write)*, 18
pcloud.encodelink *(encoding)*, 6
pcloud.encodelink(), *16*, *17*
pcloud.endpoint *(endpoint)*, 8
pcloud.endpoint(), *15*, *17*
pcloud.exists.file *(tests)*, 20
pcloud.exists.folder *(tests)*, 20
pcloud.exists.path *(tests)*, 20
pcloud.folder.empty *(tests)*, 20
pcloud.from.pctime *(time)*, 21
pcloud.fsize *(read-write)*, 18
pcloud.lapi *(pcloud-api)*, 11
pcloud.lapi(), *2*
pcloud.listfolder *(folders)*, 8
pcloud.listfolderfileids.rec *(folders)*, 8
pcloud.listfolderfiles *(folders)*, 8
pcloud.narrow *(folders)*, 8
pcloud.pub.down.filelink *(public-downloads)*, 14
pcloud.pub.down.filelink(), *7*
pcloud.pupload.data *(public-uploads)*, 16
pcloud.pupload.data.dec *(encoding)*, 6
pcloud.pupload.file *(public-uploads)*, 16
pcloud.pupload.file.dec *(encoding)*, 6
pcloud.pupload.info *(public-uploads)*, 16
pcloud.pupload.info.dec *(encoding)*, 6
pcloud.read *(read-write)*, 18
pcloud.read.link *(public-downloads)*, 14
pcloud.read.link.dec *(encoding)*, 6

pcloud.renamedir *(copy)*, 4
pcloud.renamefile *(copy)*, 4
pcloud.resp, 13
pcloud.resp(), *10*, *11*
pcloud.root *(folders)*, 8
pcloud.to.pctime *(time)*, 21
pcloud.try, 14
pcloud.try(), *10*, *11*
pcloud.upload *(read-write)*, 18
pcloud.url.code *(encoding)*, 6
pcloud.write *(read-write)*, 18
pcloudr *(pcloudr-package)*, 2
pcloudr-package, 2
public-downloads, 14
public-uploads, 16

read-write, 18
remote-paths, 19

style guide, *12*

tests, 20
time, 21