Package 'zen4R'

June 17, 2022

Version 0.6-1

Date 2022-06-17

Title Interface to 'Zenodo' REST API

Maintainer Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Depends R (>= 3.3.0), methods

Imports R6, httr, jsonlite, XML, xml2, keyring, tools, atom4R

Suggests testthat, parallel, knitr, markdown

Description Provides an Interface to 'Zenodo' (<https://zenodo.org>) REST API, including management of depositions, attribution of DOIs by 'Zenodo' and upload and download of files.

License MIT + file LICENSE

URL https://github.com/eblondel/zen4R

BugReports https://github.com/eblondel/zen4R/issues

LazyLoad yes

RoxygenNote 7.1.1

Encoding UTF-8

VignetteBuilder knitr

NeedsCompilation no

Author Emmanuel Blondel [aut, cre] (<https://orcid.org/0000-0002-5870-5762>), Julien Barde [ctb] (<https://orcid.org/0000-0002-3519-6141>), Stephen Eglen [ctb] (<https://orcid.org/0000-0001-8607-8025>), Hans Van Calster [ctb] (<https://orcid.org/0000-0001-8595-8426>), Floris Vanderhaeghe [ctb] (<https://orcid.org/0000-0002-6378-6229>)

Repository CRAN

Date/Publication 2022-06-17 11:10:02 UTC

R topics documented:

`download_zenodo` *download_zenodo*

Description

`download_zenodo` allows to download archives attached to a Zenodo record, identified by its DOI or concept DOI.

Usage

```
download_zenodo(
  doi,
  path = ".",
  files = list(),
  logger = NULL,
  quiet = FALSE,
  ...
)
```

Arguments

`doi`	a Zenodo DOI or concept DOI
`path`	the target directory where to download files
`files`	subset of filenames to restrain to download. If ignored, all files will be downloaded.
`logger`	a logger to print Zenodo API-related messages. The logger can be either NULL, "INFO" (with minimum logs), or "DEBUG" (for complete curl http calls logs)
`quiet`	Logical (`FALSE` by default). Do you want to suppress informative messages (not warnings)?
`...`	any other arguments for parallel downloading (more information at ZenodoRecord, `downloadFiles()` documentation)

Examples

```
## Not run:
 #simple download (sequential)
 download_zenodo("10.5281/zenodo.2547036")

 library(parallel)
 #download files as parallel using a cluster approach (for both Unix/Win systems)
 download_zenodo("10.5281/zenodo.2547036",
   parallel = TRUE, parallel_handler = parLapply, cl = makeCluster(2))

 #download files as parallel using mclapply (for Unix systems)
 download_zenodo("10.5281/zenodo.2547036",
   parallel = TRUE, parallel_handler = mclapply, mc.cores = 2)

## End(Not run)
```

`export_zenodo` *export_zenodo*

Description

`export_zenodo` allows to export a Zenodo record, identified by its DOI or concept DOI, using one of the export formats supported by Zenodo.

Usage

```
export_zenodo(doi, filename, format, append_format = TRUE, logger = NULL)
```

Arguments

`doi`	a Zenodo DOI or concept DOI
`filename`	a base file name (without file extension) to export to.
`format`	a valid Zenodo export format among the following: BibTeX, CSL, DataCite, DublinCore, DCAT, JSON, JSON-LD, GeoJSON, MARCXML.
`append_format`	wether format name has to be appended to the filename. Default is `TRUE` (for backward compatibility reasons). Set it to `FALSE` if you want to use only the `filename`.
`logger`	a logger to print Zenodo API-related messages. The logger can be either NULL, "INFO" (with minimum logs), or "DEBUG" (for complete curl http calls logs)

Value

the exported file name (with extension)

Examples

```
## Not run:
 export_zenodo("10.5281/zenodo.2547036", filename = "test", format = "BibTeX", append_format = F)

## End(Not run)
```

`get_versions` *get_versions*

Description

`get_versions` allows to execute a workflow

Usage

```
get_versions(doi, logger = NULL)
```

Arguments

`doi`	a Zenodo DOI or concept DOI
`logger`	a logger to print messages. The logger can be either NULL, "INFO" (with minimum logs), or "DEBUG" (for complete curl http calls logs)

Value

an object of class `data.frame` giving the record versions including date, version number and version-specific DOI.

Examples

```
## Not run:
get_versions("10.5281/zenodo.2547036")

## End(Not run)
```

`zen4R` *Interface to 'Zenodo' REST API*

Description

Provides an Interface to 'Zenodo' (<https://zenodo.org>) REST API, including management of depositions, attribution of DOIs by 'Zenodo', upload and download of files.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

zen4RLogger *zen4RLogger*

Description

zen4RLogger

zen4RLogger

Format

R6Class object.

Value

Object of R6Class for modelling a simple logger

Public fields

`verbose.info` logger info status

`verbose.debug` logger debug status

`loggerType` Logger type, either "INFO", "DEBUG" or NULL (if no logger)

Methods

Public methods:

- `zen4RLogger$logger()`
- `zen4RLogger$INFO()`
- `zen4RLogger$WARN()`
- `zen4RLogger$ERROR()`
- `zen4RLogger$new()`
- `zen4RLogger$getClassName()`
- `zen4RLogger$getClass()`
- `zen4RLogger$clone()`

Method `logger()`**:** internal logger function for the Zenodo manager

Usage:

```
zen4RLogger$logger(type, text)
```

Arguments:

`type` logger message type, "INFO", "WARN", or "ERROR"

`text` log message

Method `INFO()`**:** internal INFO logger function

Usage:

```
zen4RLogger$INFO(text)
```

Arguments:

`text` log message

Method `WARN()`**:** internal WARN logger function

Usage:

```
zen4RLogger$WARN(text)
```

Arguments:

`text` log message

Method `ERROR()`**:** internal ERROR logger function

Usage:

```
zen4RLogger$ERROR(text)
```

Arguments:

`text` log message

Method `new()`**:** initialize the Zenodo logger

Usage:

```
zen4RLogger$new(logger = NULL)
```

Arguments:

`logger` logger type NULL, 'INFO', or 'DEBUG'

Method `getClassName()`**:** Get object class name

Usage:

```
zen4RLogger$getClassName()
```

Returns: the class name, object of class `character`

Method `getClass()`**:** Get object class

Usage:

```
zen4RLogger$getClass()
```

Returns: the class, object of class `R6`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
zen4RLogger$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

Logger class used internally by zen4R

ZenodoManager *ZenodoManager*

Description

ZenodoManager

ZenodoManager

Format

R6Class object.

Value

Object of R6Class for modelling an ZenodoManager

Super class

zen4R::zen4RLogger -> ZenodoManager

Public fields

anonymous Zenodo manager anonymous status, TRUE when no token is specified

Methods

Public methods:

- ZenodoManager$new()
- ZenodoManager$getToken()
- ZenodoManager$getLicenses()
- ZenodoManager$getLicenseById()
- ZenodoManager$getCommunities()
- ZenodoManager$getCommunityById()
- ZenodoManager$getGrants()
- ZenodoManager$getGrantById()
- ZenodoManager$getFunders()
- ZenodoManager$getFunderById()
- ZenodoManager$getDepositions()
- ZenodoManager$getDepositionByConceptDOI()
- ZenodoManager$getDepositionByDOI()
- ZenodoManager$getDepositionById()
- ZenodoManager$getDepositionByConceptId()
- ZenodoManager$depositRecord()
- ZenodoManager$depositRecordVersion()

- ZenodoManager$deleteRecord()
- ZenodoManager$deleteRecordByDOI()
- ZenodoManager$deleteRecords()
- ZenodoManager$createEmptyRecord()
- ZenodoManager$editRecord()
- ZenodoManager$discardChanges()
- ZenodoManager$publishRecord()
- ZenodoManager$getFiles()
- ZenodoManager$uploadFile()
- ZenodoManager$deleteFile()
- ZenodoManager$getRecords()
- ZenodoManager$getRecordByConceptDOI()
- ZenodoManager$getRecordByDOI()
- ZenodoManager$getRecordById()
- ZenodoManager$getRecordByConceptId()
- ZenodoManager$clone()

Method `new()`**:** initializes the Zenodo Manager

Usage:

```
ZenodoManager$new(
  url = "https://zenodo.org/api",
  token = zenodo_pat(),
  logger = NULL,
  keyring_backend = "env"
)
```

Arguments:

`url` Zenodo API URL. By default, the url is set to "https://zenodo.org/api". For tests, the Zenodo sandbox API URL can be used: https://sandbox.zenodo.org/api

`token` the user token. By default an attempt will be made to retrieve token using zenodo_pat

`logger` logger type. The logger can be either NULL, "INFO" (with minimum logs), or "DEBUG" (for complete curl http calls logs)

`keyring_backend` The **keyring** backend used to store user token. The `keyring_backend` can be set to use a different backend for storing the Zenodo token with **keyring** (Default value is 'env').

Method `getToken()`**:** Get user token

Usage:

```
ZenodoManager$getToken()
```

Returns: the token, object of class `character`

Method `getLicenses()`**:** Get Licenses supported by Zenodo.

Usage:

```
ZenodoManager$getLicenses(pretty = TRUE)
```

Arguments:

`pretty` Prettify the output. By default the argument `pretty` is set to `TRUE` which will returns the list of licenses as `data.frame`. Set `pretty = FALSE` to get the raw list of licenses.

Returns: list of licenses as `data.frame` or `list`

Method `getLicenseById()`**:** Get license by Id.

Usage:

```
ZenodoManager$getLicenseById(id)
```

Arguments:

`id` license id

Returns: the license

Method `getCommunities()`**:** Get Communities supported by Zenodo.

Usage:

```
ZenodoManager$getCommunities(pretty = TRUE)
```

Arguments:

`pretty` Prettify the output. By default the argument `pretty` is set to `TRUE` which will returns the list of communities as `data.frame`. Set `pretty = FALSE` to get the raw list of communities

Returns: list of communities as `data.frame` or `list`

Method `getCommunityById()`**:** Get community by Id.

Usage:

```
ZenodoManager$getCommunityById(id)
```

Arguments:

`id` community id

Returns: the community

Method `getGrants()`**:** Get Grants supported by Zenodo.

Usage:

```
ZenodoManager$getGrants(pretty = TRUE, size = 1000)
```

Arguments:

`pretty` Prettify the output. By default the argument `pretty` is set to `TRUE` which will returns the list of grants as `data.frame`. Set `pretty = FALSE` to get the raw list of grants

`size` number of grants to be returned. By default equal to 1000.

Returns: list of grants as `data.frame` or `list`

Method `getGrantById()`**:** Get grant by Id.

Usage:

```
ZenodoManager$getGrantById(id)
```

Arguments:

`id` grant id

Returns: the grant

Method `getFunders()`**:** Get Funders supported by Zenodo.

Usage:

```
ZenodoManager$getFunders(pretty = TRUE, size = 1000)
```

Arguments:

`pretty` Prettify the output. By default the argument `pretty` is set to `TRUE` which will returns the list of funders as `data.frame`. Set `pretty = FALSE` to get the raw list of grants

`size` number of funders to be returned. By default equal to 1000.

Returns: list of funders as `data.frame` or `list`

Method `getFunderById()`**:** Get funder by Id.

Usage:

```
ZenodoManager$getFunderById(id)
```

Arguments:

`id` funder id

Returns: the funder

Method `getDepositions()`**:** Get the list of Zenodo records deposited in your Zenodo workspace. By defaut the list of depositions will be returned by page with a size of 10 results per page (default size of the Zenodo API). The parameter `q` allows to specify an ElasticSearch-compliant query to filter depositions (default query is empty to retrieve all records). The argument `all_versions`, if set to TRUE allows to get all versions of records as part of the depositions list. The argument `exact` specifies that an exact matching is wished, in which case paginated search will be disabled (only the first search page will be returned). Examples of ElasticSearch queries for Zenodo can be found at https://help.zenodo.org/guides/search/.

Usage:

```
ZenodoManager$getDepositions(
  q = "",
  size = 10,
  all_versions = FALSE,
  exact = TRUE,
  quiet = FALSE
)
```

Arguments:

`q` Elastic-Search-compliant query, as object of class `character`. Default is ""

`size` number of depositions to be retrieved per request (paginated). Default is 10

`all_versions` object of class `logical` indicating if all versions of deposits have to be retrieved. Default is `FALSE`

`exact` object of class `logical` indicating if exact matching has to be applied. Default is `TRUE`

`quiet` object of class `logical` indicating if logs have to skipped. Default is `FALSE`

Returns: a list of `ZenodoRecord`

Method `getDepositionByConceptDOI()`**:** Get a Zenodo deposition record by concept DOI (generic DOI common to all deposition record versions).

Usage:

```
ZenodoManager$getDepositionByConceptDOI(conceptdoi)
```

Arguments:

`conceptdoi` the concept DOI, object of class `character`

Returns: an object of class `ZenodoRecord` if record does exist, NULL otherwise

Method `getDepositionByDOI()`**:** Get a Zenodo deposition record by DOI.

Usage:

```
ZenodoManager$getDepositionByDOI(doi)
```

Arguments:

`doi` the DOI, object of class `character`

Returns: an object of class `ZenodoRecord` if record does exist, NULL otherwise

Method `getDepositionById()`**:** Get a Zenodo deposition record by ID.

Usage:

```
ZenodoManager$getDepositionById(recid)
```

Arguments:

`recid` the record ID, object of class `character`

Returns: an object of class `ZenodoRecord` if record does exist, NULL otherwise

Method `getDepositionByConceptId()`**:** Get a Zenodo deposition record by concept ID.

Usage:

```
ZenodoManager$getDepositionByConceptId(conceptrecid)
```

Arguments:

`conceptrecid` the record concept ID, object of class `character`

Returns: an object of class `ZenodoRecord` if record does exist, NULL otherwise

Method `depositRecord()`**:** Deposits a record on Zenodo.

Usage:

```
ZenodoManager$depositRecord(record, publish = FALSE)
```

Arguments:

`record` the record to deposit, object of class `ZenodoRecord`

`publish` object of class `logical` indicating if record has to be published (default `FALSE`). Can be set to `TRUE` (to use CAUTIOUSLY, only if you want to publish your record)

Returns: `TRUE` if deposited (and eventually published), `FALSE` otherwise

Method `depositRecordVersion()`**:** Deposits a record version on Zenodo. For details about the behavior of this function, see https://developers.zenodo.org/#new-version

Usage:

```
ZenodoManager$depositRecordVersion(
  record,
  delete_latest_files = TRUE,
  files = list(),
  publish = FALSE
)
```

Arguments:

`record` the record version to deposit, object of class `ZenodoRecord`

`delete_latest_files` object of class `logical` indicating if latest files have to be deleted. Default is `TRUE`

`files` a list of files to be uploaded with the new record version

`publish` object of class `logical` indicating if record has to be published (default `FALSE`)

Returns: `TRUE` if deposited (and eventually published), `FALSE` otherwise

Method `deleteRecord()`**:** Deletes a record given its ID

Usage:

```
ZenodoManager$deleteRecord(recordId)
```

Arguments:

`recordId` the ID of the record to be deleted

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `deleteRecordByDOI()`**:** Deletes a record by DOI

Usage:

```
ZenodoManager$deleteRecordByDOI(doi)
```

Arguments:

`doi` the DOI of the record to be deleted

Returns: `TRUE` if deleted, `FALSE` otherwise

Method `deleteRecords()`**:** Deletes all Zenodo deposited (unpublished) records. The parameter `q` allows to specify an ElasticSearch-compliant query to filter depositions (default query is empty to retrieve all records). Examples of ElasticSearch queries for Zenodo can be found at https://help.zenodo.org/guides/search/.

Usage:

```
ZenodoManager$deleteRecords(q = "", size = 10)
```

Arguments:

`q` an ElasticSearch compliant query, object of class `character`

`size` number of records to be passed to `$getDepositions` method

Returns: `TRUE` if all records have been deleted, `FALSE` otherwise

Method `createEmptyRecord()`**:** Creates an empty record in the Zenodo deposit. Returns the record newly created in Zenodo, as an object of class `ZenodoRecord` with an assigned identifier.

Usage:

```
ZenodoManager$createEmptyRecord()
```

Returns: an object of class `ZenodoRecord`

Method `editRecord()`**:** Unlocks a record already submitted. Required to edit metadata of a Zenodo record already published.

Usage:

```
ZenodoManager$editRecord(recordId)
```

Arguments:

`recordId` the ID of the record to unlock and set in editing mode.

Returns: an object of class `ZenodoRecord`

Method `discardChanges()`**:** Discards changes on a Zenodo record.

Usage:

```
ZenodoManager$discardChanges(recordId)
```

Arguments:

`recordId` the ID of the record for which changes have to be discarded.

Returns: an object of class `ZenodoRecord`

Method `publishRecord()`**:** Publishes a Zenodo record.

Usage:

```
ZenodoManager$publishRecord(recordId)
```

Arguments:

`recordId` the ID of the record to be published.

Returns: an object of class `ZenodoRecord`

Method `getFiles()`**:** Get list of files attached to a Zenodo record.

Usage:

```
ZenodoManager$getFiles(recordId)
```

Arguments:

`recordId` the ID of the record.

Returns: list of files

Method `uploadFile()`**:** Uploads a file to a Zenodo record

Usage:

```
ZenodoManager$uploadFile(path, record = NULL, recordId = NULL)
```

Arguments:

`path` Local path of the file

`record` object of class `ZenodoRecord`

`recordId` ID of the record. Deprecated, use `record` instead to take advantage of the new Zenodo bucket upload API.

Method `deleteFile()`**:** Deletes a file for a record

Usage:

```
ZenodoManager$deleteFile(recordId, fileId)
```

Arguments:

`recordId` ID of the record

`fileId` ID of the file to delete

Method `getRecords()`: Get the list of Zenodo records. By defaut the list of records will be returned by page with a size of 10 results per page (default size of the Zenodo API). The parameter `q` allows to specify an ElasticSearch-compliant query to filter depositions (default query is empty to retrieve all records). The argument `all_versions`, if set to TRUE allows to get all versions of records as part of the depositions list. The argument `exact` specifies that an exact matching is wished, in which case paginated search will be disabled (only the first search page will be returned). Examples of ElasticSearch queries for Zenodo can be found at https://help.zenodo.org/guides/search/.

Usage:

```
ZenodoManager$getRecords(
  q = "",
  size = 10,
  all_versions = FALSE,
  exact = FALSE
)
```

Arguments:

`q` Elastic-Search-compliant query, as object of class `character`. Default is ""

`size` number of records to be retrieved per request (paginated). Default is 10

`all_versions` object of class `logical` indicating if all versions of records have to be retrieved. Default is `FALSE`

`exact` object of class `logical` indicating if exact matching has to be applied. Default is `TRUE`

`quiet` object of class `logical` indicating if logs have to skipped. Default is `FALSE`

Returns: a list of `ZenodoRecord`

Method `getRecordByConceptDOI()`: Get Record by concept DOI

Usage:

```
ZenodoManager$getRecordByConceptDOI(conceptdoi)
```

Arguments:

`conceptdoi` the concept DOI

Returns: a object of class `ZenodoRecord`

Method `getRecordByDOI()`: Get Record by DOI

Usage:

```
ZenodoManager$getRecordByDOI(doi)
```

Arguments:

`doi` the DOI

Returns: a object of class `ZenodoRecord`

Method `getRecordById()`: Get Record by ID

Usage:

```
ZenodoManager$getRecordById(recid)
```

Arguments:

`recid` the record ID

Returns: a object of class ZenodoRecord

Method `getRecordByConceptId()`**:** Get Record by concept ID

Usage:

```
ZenodoManager$getRecordByConceptId(conceptrecid)
```

Arguments:

`conceptrecid` the concept ID

Returns: a object of class ZenodoRecord

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ZenodoManager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

Main user class to be used with **zen4R**

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

Examples

```
## Not run:
  ZENODO <- ZenodoManager$new(
     url = "https://sandbox.zenodo.org/api",
     token = "<your_token>",
     logger = "INFO"
  )

  #create (deposit) an empty record
  newRec <- ZENODO$createEmptyRecord()

  #create and fill a local (not yet deposited) record
  myrec <- ZenodoRecord$new()
  myrec$setTitle("my R package")
  myrec$setDescription("A description of my R package")
  myrec$setUploadType("software")
  myrec$addCreator(
    firstname = "John", lastname = "Doe",
    affiliation = "Independent", orcid = "0000-0000-0000-0000"
   )
  myrec$setLicense("mit")
  myrec$setAccessRight("open")
 myrec$setDOI("mydoi") #use this method if your DOI has been assigned elsewhere, outside Zenodo
  myrec$addCommunity("ecfunded")
```

```
#deposit the record
myrec <- ZENODO$depositRecord(myrec)

#publish a record (with caution!!)
#this method will PUBLISH the deposition done earlier
ZENODO$publishRecord(myrec$id)
#With even more caution the publication can be done with a shortcut argument at deposit time
ZENODO$depositRecord(myrec, publish = TRUE)

#delete a record (by id)
#this methods only works for unpublished deposits
#(if a record is published, it cannot be deleted anymore!)
ZENODO$deleteRecord(myrec$id)

#HOW TO UPLOAD FILES to a deposit

#upload a file
ZENODO$uploadFile("path/to/your/file", record = myrec)

#list files
zen_files <- ZENODO$getFiles(myrec$id)

#delete a file?
ZENODO$deleteFile(myrec$id, zen_files[[1]]$id)

## End(Not run)
```

ZenodoRecord *ZenodoRecord*

Description

ZenodoRecord

ZenodoRecord

Format

R6Class object.

Value

Object of R6Class for modelling an ZenodoRecord

Super class

zen4R::zen4RLogger -> ZenodoRecord

Public fields

`conceptdoi` record Concept DOI (common to all record versions)

`conceptrecid` record concept id

`created` record creation date

`doi` record doi

`doi_url` record doi URL

`files` list of files associated to the record

`id` record id

`links` list of links associated to the record

`metadata` metadata elements associated to the record

`modified` record modification date

`owner` record owner

`record_id` record_id

`state` record state

`submitted` record submission status

`title` record title

`version` record version

Methods

Public methods:

- `ZenodoRecord$new()`
- `ZenodoRecord$prereserveDOI()`
- `ZenodoRecord$setDOI()`
- `ZenodoRecord$getConceptDOI()`
- `ZenodoRecord$getFirstDOI()`
- `ZenodoRecord$getLastDOI()`
- `ZenodoRecord$getVersions()`
- `ZenodoRecord$setUploadType()`
- `ZenodoRecord$setPublicationType()`
- `ZenodoRecord$setImageType()`
- `ZenodoRecord$setPublicationDate()`
- `ZenodoRecord$setEmbargoDate()`
- `ZenodoRecord$setTitle()`
- `ZenodoRecord$setDescription()`
- `ZenodoRecord$setAccessRight()`
- `ZenodoRecord$setAccessConditions()`
- `ZenodoRecord$addCreator()`
- `ZenodoRecord$removeCreator()`
- `ZenodoRecord$removeCreatorByName()`

- ZenodoRecord$removeCreatorByAffiliation()
- ZenodoRecord$removeCreatorByORCID()
- ZenodoRecord$removeCreatorByGND()
- ZenodoRecord$addContributor()
- ZenodoRecord$removeContributor()
- ZenodoRecord$removeContributorByName()
- ZenodoRecord$removeContributorByAffiliation()
- ZenodoRecord$removeContributorByORCID()
- ZenodoRecord$removeContributorByGND()
- ZenodoRecord$setLicense()
- ZenodoRecord$setVersion()
- ZenodoRecord$setLanguage()
- ZenodoRecord$addRelatedIdentifier()
- ZenodoRecord$removeRelatedIdentifier()
- ZenodoRecord$setReferences()
- ZenodoRecord$addReference()
- ZenodoRecord$removeReference()
- ZenodoRecord$setKeywords()
- ZenodoRecord$addKeyword()
- ZenodoRecord$removeKeyword()
- ZenodoRecord$addSubject()
- ZenodoRecord$removeSubject()
- ZenodoRecord$removeSubjectByTerm()
- ZenodoRecord$removeSubjectByIdentifier()
- ZenodoRecord$setNotes()
- ZenodoRecord$setCommunities()
- ZenodoRecord$addCommunity()
- ZenodoRecord$removeCommunity()
- ZenodoRecord$setGrants()
- ZenodoRecord$addGrant()
- ZenodoRecord$removeGrant()
- ZenodoRecord$setJournalTitle()
- ZenodoRecord$setJournalVolume()
- ZenodoRecord$setJournalIssue()
- ZenodoRecord$setJournalPages()
- ZenodoRecord$setConferenceTitle()
- ZenodoRecord$setConferenceAcronym()
- ZenodoRecord$setConferenceDates()
- ZenodoRecord$setConferencePlace()
- ZenodoRecord$setConferenceUrl()
- ZenodoRecord$setConferenceSession()
- ZenodoRecord$setConferenceSessionPart()

- ZenodoRecord$setImprintPublisher()
- ZenodoRecord$setImprintISBN()
- ZenodoRecord$setImprintPlace()
- ZenodoRecord$setPartofTitle()
- ZenodoRecord$setPartofPages()
- ZenodoRecord$setThesisUniversity()
- ZenodoRecord$addThesisSupervisor()
- ZenodoRecord$removeThesisSupervisor()
- ZenodoRecord$removeThesisSupervisorByName()
- ZenodoRecord$removeThesisSupervisorByAffiliation()
- ZenodoRecord$removeThesisSupervisorByORCID()
- ZenodoRecord$removeThesisSupervisorByGND()
- ZenodoRecord$exportAs()
- ZenodoRecord$exportAsBibTeX()
- ZenodoRecord$exportAsCSL()
- ZenodoRecord$exportAsDataCite()
- ZenodoRecord$exportAsDublinCore()
- ZenodoRecord$exportAsDCAT()
- ZenodoRecord$exportAsJSON()
- ZenodoRecord$exportAsJSONLD()
- ZenodoRecord$exportAsGeoJSON()
- ZenodoRecord$exportAsMARCXML()
- ZenodoRecord$exportAsAllFormats()
- ZenodoRecord$listFiles()
- ZenodoRecord$downloadFiles()
- ZenodoRecord$print()
- ZenodoRecord$toDCEntry()
- ZenodoRecord$clone()

Method `new()`**:** method is used to instantiate a ZenodoRecord

Usage:

```
ZenodoRecord$new(obj = NULL, logger = "INFO")
```

Arguments:

`obj` an optional list object to create the record

`logger` a logger to print log messages. It can be either NULL, "INFO" (with minimum logs), or "DEBUG" (for complete curl http calls logs)

Method `prereserveDOI()`**:** Set prereserve_doi if `TRUE`, `FALSE` otherwise to create a record without prereserved DOI by Zenodo. By default, this method will be called to prereserve a DOI assuming the record created doesn't yet handle a DOI. To avoid prereserving a DOI call `$prereserveDOI(FALSE)` on your record.

Usage:

```
ZenodoRecord$prereserveDOI(prereserve)
```

Arguments:

`prereserve` whether a DOI has to be pre-reserved by Zenodo

Method `setDOI()`**:** Set the DOI. This method can be used if a DOI has been already assigned outside Zenodo. This method will call the method `$prereserveDOI(FALSE)`.

Usage:

```
ZenodoRecord$setDOI(doi)
```

Arguments:

`doi` DOI to set for the record

Method `getConceptDOI()`**:** Get the concept (generic) DOI. The concept DOI is a generic DOI common to all versions of a Zenodo record. When a deposit is unsubmitted, this concept DOI is inherited based on the prereserved DOI of the first record version.

Usage:

```
ZenodoRecord$getConceptDOI()
```

Returns: the concept DOI, object of class `character`

Method `getFirstDOI()`**:** Get DOI of the first record version.

Usage:

```
ZenodoRecord$getFirstDOI()
```

Returns: the first DOI, object of class `character`

Method `getLastDOI()`**:** Get DOI of the latest record version.

Usage:

```
ZenodoRecord$getLastDOI()
```

Returns: the last DOI, object of class `character`

Method `getVersions()`**:** Get record versions with creation/publication date, version (ordering number) and DOI.

Usage:

```
ZenodoRecord$getVersions()
```

Returns: a `data.frame` with the record versions

Method `setUploadType()`**:** Set the upload type (mandatory).

Usage:

```
ZenodoRecord$setUploadType(uploadType)
```

Arguments:

`uploadType` record upload type among the following values: 'publication', 'poster','presentation','dataset','image','vide
or 'software'

Method `setPublicationType()`**:** Set the publication type (mandatory if upload type is 'publication').

Usage:

```
ZenodoRecord$setPublicationType(publicationType)
```

Arguments:

`publicationType` record publication type among the following values: 'book','section', 'conferencepaper','article','patent','preprint','report','softwaredocumentation','thesis', 'technicalnote','workingpaper', or 'other'

Method `setImageType()`**:** Set the image type (mandatory if image type is 'image').

Usage:

```
ZenodoRecord$setImageType(imageType)
```

Arguments:

`imageType` record publication type among the following values: 'figure','plot', 'drawing','diagram','photo', or 'other'

Method `setPublicationDate()`**:** Set the publication date.

Usage:

```
ZenodoRecord$setPublicationDate(publicationDate)
```

Arguments:

`publicationDate` object of class `Date`

Method `setEmbargoDate()`**:** Set the embargo date.

Usage:

```
ZenodoRecord$setEmbargoDate(embargoDate)
```

Arguments:

`embargoDate` object of class `Date`

Method `setTitle()`**:** Set the record title.

Usage:

```
ZenodoRecord$setTitle(title)
```

Arguments:

`title` object of class `character`

Method `setDescription()`**:** Set the record description

Usage:

```
ZenodoRecord$setDescription(description)
```

Arguments:

`description` object of class `character`

Method `setAccessRight()`**:** Set the access right.

Usage:

```
ZenodoRecord$setAccessRight(accessRight)
```

Arguments:

`accessRight` record access right among the following values: 'open','embargoed', 'restricted','closed'

Method `setAccessConditions()`**:** set the access conditions.

Usage:

```
ZenodoRecord$setAccessConditions(accessConditions)
```

Arguments:

`accessConditions` object of class `character`

Method `addCreator()`**:** Add a creator for the record. One approach is to use the `firstname` and `lastname` arguments, that by default will be concatenated for Zenodo as `lastname, firstname`. For more flexibility over this, the `name` argument can be directly used.

Usage:

```
ZenodoRecord$addCreator(
  firstname,
  lastname,
  name = paste(lastname, firstname, sep = ", "),
  affiliation = NULL,
  orcid = NULL,
  gnd = NULL
)
```

Arguments:

`firstname` creator first name

`lastname` creator last name

`name` creator name

`affiliation` creator affiliation (optional)

`orcid` creator ORCID (optional)

`gnd` creator GND (optional)

Returns: `TRUE` if added, `FALSE` otherwise

Method `removeCreator()`**:** Removes a creator by a property. The `by` parameter should be the name of the creator property ('name' - in the form 'lastname, firstname', 'affiliation', 'orcid' or 'gnd').

Usage:

```
ZenodoRecord$removeCreator(by, property)
```

Arguments:

`by` property used as criterion to remove the creator

`property` property value used to remove the creator

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeCreatorByName()`**:** Removes a creator by name.

Usage:

```
ZenodoRecord$removeCreatorByName(name)
```

Arguments:

`name` creator name

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeCreatorByAffiliation()`**:** Removes a creator by affiliation.

Usage:

```
ZenodoRecord$removeCreatorByAffiliation(affiliation)
```

Arguments:

`affiliation` creator affiliation

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeCreatorByORCID()`**:** Removes a creator by ORCID.

Usage:

```
ZenodoRecord$removeCreatorByORCID(orcid)
```

Arguments:

`orcid` creator ORCID

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeCreatorByGND()`**:** Removes a creator by GND.

Usage:

```
ZenodoRecord$removeCreatorByGND(gnd)
```

Arguments:

`gnd` creator GND

Returns: `TRUE` if removed, `FALSE` otherwise

Method `addContributor()`**:** Add a contributor for the record. Firstname, lastname, and type are mandatory.

Usage:

```
ZenodoRecord$addContributor(
  firstname,
  lastname,
  type,
  affiliation = NULL,
  orcid = NULL,
  gnd = NULL
)
```

Arguments:

`firstname` contributor first name

`lastname` contributor last name

`type` contributor type, among values: ContactPerson, DataCollector, DataCurator, DataManager, Distributor, Editor, Funder, HostingInstitution, Producer, ProjectLeader, ProjectManager, ProjectMember, RegistrationAgency, RegistrationAuthority, RelatedPerson, Researcher, ResearchGroup, RightsHolder, Supervisor, Sponsor, WorkPackageLeader, Other.

`affiliation` contributor affiliation (optional)

`orcid` contributor orcid (optional)

`gnd` contributor gnd (optional)

Returns: `TRUE` if added, `FALSE` otherwise

Method `removeContributor()`**:** Removes a contributor by a property. The by parameter should be the name of the contributor property ('name' - in the form 'lastname, firstname', 'affiliation', 'orcid' or 'gnd'). `FALSE` otherwise.

Usage:

```
ZenodoRecord$removeContributor(by, property)
```

Arguments:

`by` property used as criterion to remove the contributor

`property` property value used to remove the contributor

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeContributorByName()`**:** Removes a contributor by name.

Usage:

```
ZenodoRecord$removeContributorByName(name)
```

Arguments:

`name` contributor name

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeContributorByAffiliation()`**:** Removes a contributor by affiliation.

Usage:

```
ZenodoRecord$removeContributorByAffiliation(affiliation)
```

Arguments:

`affiliation` contributor affiliation

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeContributorByORCID()`**:** Removes a contributor by ORCID.

Usage:

```
ZenodoRecord$removeContributorByORCID(orcid)
```

Arguments:

`orcid` contributor ORCID

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeContributorByGND()`**:** Removes a contributor by GND.

Usage:

```
ZenodoRecord$removeContributorByGND(gnd)
```

Arguments:

`gnd` contributor GND

Returns: `TRUE` if removed, `FALSE` otherwise

Method `setLicense()`**:** Set license. The license should be set with the Zenodo id of the license. If not recognized by Zenodo, the function will return an error. The list of licenses can fetched with the `ZenodoManager` and the function `$getLicenses()`.

Usage:

```
ZenodoRecord$setLicense(licenseId)
```

Arguments:

`licenseId` a license Id

Method `setVersion()`**:** Set record version.

Usage:

```
ZenodoRecord$setVersion(version)
```

Arguments:

`version` the record version to set

Method `setLanguage()`**:** Set the language.

Usage:

```
ZenodoRecord$setLanguage(language)
```

Arguments:

`language` ISO 639-2 or 639-3 code

Method `addRelatedIdentifier()`**:** Adds a related identifier with a given relation.

Usage:

```
ZenodoRecord$addRelatedIdentifier(relation, identifier)
```

Arguments:

`relation` relation type among following values: isCitedBy, cites, isSupplementTo, isSupplementedBy, isNewVersionOf, isPreviousVersionOf, isPartOf, hasPart, compiles, isCompiledBy, isIdenticalTo, isAlternateIdentifier

`identifier` resource identifier

Method `removeRelatedIdentifier()`**:** Removes a related identifier with a given relation.

Usage:

```
ZenodoRecord$removeRelatedIdentifier(relation, identifier)
```

Arguments:

`relation` relation type among following values: isCitedBy, cites, isSupplementTo, isSupplementedBy, isNewVersionOf, isPreviousVersionOf, isPartOf, hasPart, compiles, isCompiledBy, isIdenticalTo, isAlternateIdentifier

`identifier` resource identifier

Method `setReferences()`**:** Set references

Usage:

```
ZenodoRecord$setReferences(references)
```

Arguments:

`references` a vector or list of references to set for the record

Method `addReference()`**:** Add a reference

Usage:

```
ZenodoRecord$addReference(reference)
```

Arguments:

`reference` the reference to add

Returns: `TRUE` if added, `FALSE` otherwise

Method `removeReference()`**:** Remove a reference

Usage:

```
ZenodoRecord$removeReference(reference)
```

Arguments:

`reference` the reference to remove

Returns: `TRUE` if removed, `FALSE` otherwise

Method `setKeywords()`**:** Set keywords

Usage:

```
ZenodoRecord$setKeywords(keywords)
```

Arguments:

`keywords` a vector or list of keywords to set for the record

Method `addKeyword()`**:** Add a keyword

Usage:

```
ZenodoRecord$addKeyword(keyword)
```

Arguments:

`keyword` the keyword to add

Returns: `TRUE` if added, `FALSE` otherwise

Method `removeKeyword()`**:** Remove a keyword

Usage:

```
ZenodoRecord$removeKeyword(keyword)
```

Arguments:

`keyword` the keyword to remove

Returns: `TRUE` if removed, `FALSE` otherwise

Method `addSubject()`**:** Adds a subject given a term and identifier

Usage:

```
ZenodoRecord$addSubject(term, identifier)
```

Arguments:

`term` subject term

`identifier` subject identifier

Method `removeSubject()`**:** Removes subject(s) by a property. The by parameter should be the name of the subject property ('term' or 'identifier').

Usage:

```
ZenodoRecord$removeSubject(by, property)
```

Arguments:

`by` property used as criterion to remove subjects

`property` property value used to remove subjects

Returns: `TRUE` if at least one subject is removed, `FALSE` otherwise.

Method `removeSubjectByTerm()`**:** Removes subject(s) by term.

Usage:

```
ZenodoRecord$removeSubjectByTerm(term)
```

Arguments:

`term` the term to use to remove subject(s)

Returns: `TRUE` if at least one subject is removed, `FALSE` otherwise.

Method `removeSubjectByIdentifier()`**:** Removes subject(s) by identifier

Usage:

```
ZenodoRecord$removeSubjectByIdentifier(identifier)
```

Arguments:

`identifier` the identifier to use to remove subject(s)

Returns: `TRUE` if at least one subject is removed, `FALSE` otherwise.

Method `setNotes()`**:** Set notes. HTML is not allowed

Usage:

```
ZenodoRecord$setNotes(notes)
```

Arguments:

`notes` object of class `character`

Method `setCommunities()`**:** Set a vector of character strings identifying communities

Usage:

```
ZenodoRecord$setCommunities(communities)
```

Arguments:

`communities` a vector or list of communities. Values should among known communities. The list of communities can fetched with the `ZenodoManager` and the function `$getCommunities()`. Each community should be set with the Zenodo id of the community. If not recognized by Zenodo, the function will return an error.

Method `addCommunity()`**:** Adds a community to the record metadata.

Usage:

```
ZenodoRecord$addCommunity(community)
```

Arguments:

`community` community to add. The community should be set with the Zenodo id of the community. If not recognized by Zenodo, the function will return an error. The list of communities can fetched with the `ZenodoManager` and the function `$getCommunities()`.

Returns: `TRUE` if added, `FALSE` otherwise

Method `removeCommunity()`**:** Removes a community from the record metadata.

Usage:

```
ZenodoRecord$removeCommunity(community)
```

Arguments:

`community` community to remove. The community should be set with the Zenodo id of the community.

Returns: `TRUE` if removed, `FALSE` otherwise

Method `setGrants()`**:** Set a vector of character strings identifying grants

Usage:

```
ZenodoRecord$setGrants(grants)
```

Arguments:

`grants` a vector or list of grants Values should among known grants The list of grants can fetched with the `ZenodoManager` and the function `$getGrants()`. Each grant should be set with the Zenodo id of the grant If not recognized by Zenodo, the function will raise a warning only.

Method `addGrant()`**:** Adds a grant to the record metadata.

Usage:

```
ZenodoRecord$addGrant(grant)
```

Arguments:

`grant` grant to add. The grant should be set with the id of the grant. If not recognized by Zenodo, the function will return an warning only. The list of grants can fetched with the `ZenodoManager` and the function `$getGrants()`.

Returns: `TRUE` if added, `FALSE` otherwise

Method `removeGrant()`**:** Removes a grant from the record metadata.

Usage:

```
ZenodoRecord$removeGrant(grant)
```

Arguments:

`grant` grant to remove. The grant should be set with the Zenodo id of the grant

Returns: `TRUE` if removed, `FALSE` otherwise

Method `setJournalTitle()`**:** Set Journal title to the record metadata

Usage:

```
ZenodoRecord$setJournalTitle(title)
```

Arguments:

`title` a title, object of class `character`

Method `setJournalVolume()`**:** Set Journal volume to the record metadata

Usage:

```
ZenodoRecord$setJournalVolume(volume)
```

Arguments:

`volume` a volume

Method `setJournalIssue()`**:** Set Journal issue to the record metadata

Usage:

```
ZenodoRecord$setJournalIssue(issue)
```

Arguments:

`issue` an issue

Method `setJournalPages()`**:** Set Journal pages to the record metadata

Usage:

```
ZenodoRecord$setJournalPages(pages)
```

Arguments:

`pages` number of pages

Method `setConferenceTitle()`**:** Set conference title to the record metadata

Usage:

```
ZenodoRecord$setConferenceTitle(title)
```

Arguments:

`title` conference title, object of class `character`

Method `setConferenceAcronym()`**:** Set conference acronym to the record metadata

Usage:

```
ZenodoRecord$setConferenceAcronym(acronym)
```

Arguments:

`acronym` conference acronym, object of class `character`

Method `setConferenceDates()`**:** Set conference dates to the record metadata

Usage:

```
ZenodoRecord$setConferenceDates(dates)
```

Arguments:

`dates` conference dates, object of class `character`

Method `setConferencePlace()`**:** Set conference place to the record metadata

Usage:

```
ZenodoRecord$setConferencePlace(place)
```

Arguments:

`place` conference place, object of class `character`

Method `setConferenceUrl()`**:** Set conference url to the record metadata

Usage:

```
ZenodoRecord$setConferenceUrl(url)
```

Arguments:

`url` conference url, object of class `character`

Method `setConferenceSession()`**:** Set conference session to the record metadata

Usage:

```
ZenodoRecord$setConferenceSession(session)
```

Arguments:

`session` conference session, object of class `character`

Method `setConferenceSessionPart()`**:** Set conference session part to the record metadata

Usage:

```
ZenodoRecord$setConferenceSessionPart(part)
```

Arguments:

`part` conference session part, object of class `character`

Method `setImprintPublisher()`**:** Set imprint publisher to the record metadata

Usage:

```
ZenodoRecord$setImprintPublisher(publisher)
```

Arguments:

`publisher` the publisher, object of class `character`

Method `setImprintISBN()`**:** Set imprint ISBN to the record metadata

Usage:

```
ZenodoRecord$setImprintISBN(isbn)
```

Arguments:

`isbn` the ISBN, object of class `character`

Method `setImprintPlace()`**:** Set imprint place to the record metadata

Usage:

```
ZenodoRecord$setImprintPlace(place)
```

Arguments:

`place` the place, object of class `character`

Method `setPartofTitle()`**:** Set title to which record is part of

Usage:

```
ZenodoRecord$setPartofTitle(title)
```

Arguments:

`title` the title, object of class `character`

Method `setPartofPages()`**:** Set pages to which record is part of

Usage:

```
ZenodoRecord$setPartofPages(pages)
```

Arguments:

`pages` the pages, object of class `character`

Method `setThesisUniversity()`**:** Set thesis university

Usage:

```
ZenodoRecord$setThesisUniversity(university)
```

Arguments:

`university` the university, object of class `character`

Method `addThesisSupervisor()`**:** Adds thesis supervisor

Usage:

```
ZenodoRecord$addThesisSupervisor(
  firstname,
  lastname,
  affiliation = NULL,
  orcid = NULL,
  gnd = NULL
)
```

Arguments:

`firstname` supervisor first name

`lastname` supervisor last name

`affiliation` supervisor affiliation (optional)

`orcid` supervisor ORCID (optional)

`gnd` supervisor GND (optional)

Method `removeThesisSupervisor()`**:** Removes a thesis supervisor by a property. The by parameter should be the name of the thesis supervisor property ('name' - in the form 'lastname, firstname', 'affiliation', 'orcid' or 'gnd').

Usage:

```
ZenodoRecord$removeThesisSupervisor(by, property)
```

Arguments:

`by` property used as criterion to remove the thesis supervisor

`property` property value used to remove the thesis supervisor

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeThesisSupervisorByName()`**:** Removes a thesis supervisor by name.

Usage:

```
ZenodoRecord$removeThesisSupervisorByName(name)
```

Arguments:

`name` thesis supervisor name

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeThesisSupervisorByAffiliation()`**:** Removes a thesis supervisor by affiliation

Usage:

```
ZenodoRecord$removeThesisSupervisorByAffiliation(affiliation)
```

Arguments:

`affiliation` thesis supervisor affiliation

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeThesisSupervisorByORCID()`**:** Removes a thesis supervisor by ORCID

Usage:

```
ZenodoRecord$removeThesisSupervisorByORCID(orcid)
```

Arguments:

`orcid` thesis supervisor ORCID

Returns: `TRUE` if removed, `FALSE` otherwise

Method `removeThesisSupervisorByGND()`**:** Removes a thesis supervisor by GND

Usage:

```
ZenodoRecord$removeThesisSupervisorByGND(gnd)
```

Arguments:

`gnd` thesis supervisor GND

Returns: `TRUE` if removed, `FALSE` otherwise

Method `exportAs()`**:** Exports record to a file by format.

Usage:

```
ZenodoRecord$exportAs(format, filename, append_format = TRUE)
```

Arguments:

`format` the export format to use. Possibles values are: BibTeX, CSL, DataCite, DublinCore, DCAT, JSON, JSON-LD, GeoJSON, MARCXML

`filename` the target filename (without extension)

`append_format` wether format name has to be appended to the filename. Default is `TRUE` (for backward compatibility reasons). Set it to `FALSE` if you want to use only the `filename`.

Returns: the writen file name (with extension)

Method `exportAsBibTeX()`**:** Exports record as BibTeX

Usage:

```
ZenodoRecord$exportAsBibTeX(filename)
```

Arguments:

`filename` the target filename (without extension)

Returns: the writen file name (with extension)

Method `exportAsCSL()`**:** Exports record as CSL

Usage:

```
ZenodoRecord$exportAsCSL(filename)
```

Arguments:

`filename` the target filename (without extension)

Returns: the writen file name (with extension)

Method `exportAsDataCite()`**:** Exports record as DataCite

Usage:

```
ZenodoRecord$exportAsDataCite(filename)
```

Arguments:

`filename` the target filename (without extension)

Returns: the writen file name (with extension)

Method `exportAsDublinCore()`**:** Exports record as DublinCore

Usage:

```
ZenodoRecord$exportAsDublinCore(filename)
```

Arguments:

`filename` the target filename (without extension)

Returns: the writen file name (with extension)

Method `exportAsDCAT()`**:** Exports record as DCAT

Usage:

```
ZenodoRecord$exportAsDCAT(filename)
```

Arguments:

`filename` the target filename (without extension)

Returns: the writen file name (with extension)

Method `exportAsJSON()`**:** Exports record as JSON

Usage:

```
ZenodoRecord$exportAsJSON(filename)
```

Arguments:

`filename` the target filename (without extension)

Returns: the writen file name (with extension)

Method `exportAsJSONLD()`**:** Exports record as JSONLD

Usage:

```
ZenodoRecord$exportAsJSONLD(filename)
```

Arguments:

`filename` the target filename (without extension)

Method `exportAsGeoJSON()`**:** Exports record as GeoJSON

Usage:

```
ZenodoRecord$exportAsGeoJSON(filename)
```

Arguments:

`filename` the target filename (without extension)

Returns: the writen file name (with extension)

Method `exportAsMARCXML()`**:** Exports record as MARCXML

Usage:

```
ZenodoRecord$exportAsMARCXML(filename)
```

Arguments:

`filename` the target filename (without extension)

Returns: the writen file name (with extension)

Method `exportAsAllFormats()`**:** Exports record in all Zenodo record export formats. This function will create one file per Zenodo metadata formats.

Usage:

```
ZenodoRecord$exportAsAllFormats(filename)
```

Arguments:

`filename` the target filename (without extension)

Method `listFiles()`**:** list files attached to the record

Usage:

```
ZenodoRecord$listFiles(pretty = TRUE)
```

Arguments:

`pretty` whether a pretty output (`data.frame`) should be returned (default `TRUE`), otherwise the raw list of files is returned.

Returns: the files, as `data.frame` or `list`

Method `downloadFiles()`**:** Downloads files attached to the record

Usage:

```
ZenodoRecord$downloadFiles(
  path = ".",
  files = list(),
  parallel = FALSE,
  parallel_handler = NULL,
  cl = NULL,
  quiet = FALSE,
  ...
)
```

Arguments:

`path` target download path (by default it will be the current working directory)

`files` (list of) file(s) to download. If not specified, by default all files will be downloaded.

`parallel` whether download has to be done in parallel using the chosen `parallel_handler`. Default is `FALSE`

`parallel_handler` The parallel handler to use eg. `mclapply`. To use a different parallel handler (such as eg `parLapply` or `parSapply`), specify its function in `parallel_handler` argument. For cluster-based parallel download, this is the way to proceed. In that case, the cluster should be created earlier by the user with `makeCluster` and passed as `cl` argument. After downloading all files, the cluster will be stopped automatically.

`cl` an optional cluster for cluster-based parallel handlers

`quiet` (default is `FALSE`) can be set to suppress informative messages (not warnings).

`...` arguments inherited from `parallel::mclapply` or the custom `parallel_handler` can be added (eg. `mc.cores` for `mclapply`)

Method `print()`**:** Prints a ZenodoRecord

Usage:

```
ZenodoRecord$print(..., format = "internal", depth = 1)
```

Arguments:

`...` any other parameter. Not used

`format` format to use for printing. By default, `internal` uses an **zen4R** internal printing method. Other methods available are those supported by Zenodo for record export, and can be used only if the record has already been published (with a DOI). Attemps to print using a Zenodo export format for a record will raise a warning message and revert to "internal" format

`depth` an internal depth parameter for indentation of print statements, in case of listing or recursive use of print

Method `toDCEntry()`**:** Maps to an **atom4R** DCEntry. Note: applies only to published records.

Usage:

```
ZenodoRecord$toDCEntry()
```

Returns: an object of class `DCEntry`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ZenodoRecord$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

See examples in `download_zenodo` utility function.

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

ZenodoRequest *ZenodoRequest*

Description

ZenodoRequest

ZenodoRequest

Format

R6Class object.

Value

Object of R6Class for modelling a generic Zenodo request

Super class

zen4R::zen4RLogger -> ZenodoRequest

Methods

Public methods:

- ZenodoRequest$new()
- ZenodoRequest$execute()
- ZenodoRequest$getRequest()
- ZenodoRequest$getRequestHeaders()
- ZenodoRequest$getStatus()
- ZenodoRequest$getResponse()
- ZenodoRequest$getException()
- ZenodoRequest$getResult()
- ZenodoRequest$setResult()
- ZenodoRequest$clone()

Method `new()`**:** Initializes a ZenodoRequest

Usage:

```
ZenodoRequest$new(
  url,
  type,
  request,
  data = NULL,
  file = NULL,
  token,
  logger = NULL,
  ...
)
```

Arguments:
`url` request URL
`type` Type of request: 'GET', 'POST', 'PUT', 'DELETE'
`request` the method request
`data` payload (optional)
`file` to be uploaded (optional)
`token` user token
`logger` the logger type
`...` any other arg

Method `execute()`**:** Executes the request
Usage:
```
ZenodoRequest$execute()
```

Method `getRequest()`**:** Get request
Usage:
```
ZenodoRequest$getRequest()
```

Method `getRequestHeaders()`**:** Get request headers
Usage:
```
ZenodoRequest$getRequestHeaders()
```

Method `getStatus()`**:** Get request status
Usage:
```
ZenodoRequest$getStatus()
```

Method `getResponse()`**:** Get request response
Usage:
```
ZenodoRequest$getResponse()
```

Method `getException()`**:** Get request exception
Usage:
```
ZenodoRequest$getException()
```

Method `getResult()`**:** Get request result
Usage:
```
ZenodoRequest$getResult()
```

Method `setResult()`**:** Set request result
Usage:
```
ZenodoRequest$setResult(result)
```
Arguments:
`result` result to be set

Method `clone()`**:** The objects of this class are cloneable with this method.
Usage:
```
ZenodoRequest$clone(deep = FALSE)
```
Arguments:
`deep` Whether to make a deep clone.

Note

Abstract class used internally by **zen4R**

Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

`zenodo_pat` *zenodo_pat*

Description

Get Zenodo personal access token, looking in env var 'ZENODO_PAT'

Usage

```
zenodo_pat(quiet = TRUE)
```

Arguments

`quiet`	Hide log message, default is `TRUE`

Index