

# Package ‘zen4R’

March 30, 2026

**Version** 0.10.5

**Date** 2026-03-29

**Title** Interface to 'Zenodo' REST API

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

**Depends** R (>= 3.3.0), methods

**Imports** R6, cli, httr, jsonlite, XML, xml2, keyring, tools, utf8, plyr

**Suggests** testthat, parallel, knitr, markdown, atom4R

**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** 7.3.3

**Encoding** UTF-8

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Emmanuel Blondel [aut, cre] (ORCID:

<<https://orcid.org/0000-0002-5870-5762>>),

Julien Barde [ctb] (ORCID: <<https://orcid.org/0000-0002-3519-6141>>),

Stephen Eglen [ctb] (ORCID: <<https://orcid.org/0000-0001-8607-8025>>),

Hans Van Calster [ctb] (ORCID: <<https://orcid.org/0000-0001-8595-8426>>),

Floris Vanderhaeghe [ctb] (ORCID:

<<https://orcid.org/0000-0002-6378-6229>>),

Jemma Stachelek [ctb] (ORCID: <<https://orcid.org/0000-0002-5924-2464>>),

Collin Schwantes [ctb] (ORCID: <<https://orcid.org/0000-0002-9882-941X>>),

Nicholas Tierney [ctb] (ORCID: <<https://orcid.org/0000-0003-1460-8722>>)

**Repository** CRAN

**Date/Publication** 2026-03-30 07:50:02 UTC

## Contents

download_zenodo . . . . .	2
export_zenodo . . . . .	3
get_citation . . . . .	4
get_licenses . . . . .	5
get_versions . . . . .	6
get_zenodo . . . . .	6
zen4R . . . . .	7
zen4RLogger . . . . .	8
ZenodoException . . . . .	10
ZenodoManager . . . . .	11
ZenodoRecord . . . . .	28
ZenodoRequest . . . . .	54
zenodo_pat . . . . .	56
<b>Index</b>	<b>57</b>

---

download_zenodo	<i>download_zenodo</i>
-----------------	------------------------

---

## 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(),
  sandbox = FALSE,
  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.
sandbox	Use the sandbox infrastructure. Default is FALSE
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 downloading (more information at <a href="#">ZenodoRecord</a> , <code>downloadFiles()</code> 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	<i>export_zenodo</i>
---------------	----------------------

---

## 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,
  sandbox = FALSE,
  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 (or DataCiteXML), DublinCore, DCAT, JSON, JSON-LD, GeoJSON, MARCXML, DataCiteJSON, CodeMeta, DataPackage, CFF.
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.
sandbox	Use the sandbox infrastructure. Default is FALSE
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)
```

---

<i>get_citation</i>	<i>get_citation</i>
---------------------	---------------------

---

**Description**

*get\_citation* allows to get the styled citation for Zenodo record, identified by its DOI or concept DOI, and the style name

**Usage**

```
get_citation(
  doi,
  sandbox = FALSE,
  style = c("havard-cite-them-right", "apa", "modern-language-association", "vancouver",
            "chicago-fullnote-bibliography", "ieee", "bibtex"),
  logger = NULL
)
```

**Arguments**

doi	a Zenodo DOI or concept DOI
sandbox	Use the sandbox infrastructure. Default is FALSE
style	the style character string among. Possible values "havard-cite-them-right", "apa", "modern-language-association", "vancouver", "chicago-fullnote-bibliography", "ieee", or "bibtex".

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.

the record citation

### Examples

```
## Not run:
  get_citation(doi = "10.5281/zenodo.2547036", style = "ieee")

## End(Not run)
```

---

get_licenses	<i>get_licenses</i>
--------------	---------------------

---

### Description

`get_licenses` allows to list all licenses supported by Zenodo.

### Usage

```
get_licenses(pretty = TRUE, sandbox = FALSE)
```

### Arguments

pretty            output delivered as `data.frame`  
sandbox           Use the sandbox infrastructure. Default is FALSE

### Value

the licenses as `list` or `data.frame`

### Examples

```
## Not run:
  get_licenses(pretty = TRUE)

## End(Not run)
```

get\_versions

*get\_versions*

---

**Description**

get\_versions allows to execute a workflow

**Usage**

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

**Arguments**

doi	a Zenodo DOI or concept DOI
sandbox	Use the sandbox infrastructure. Default is FALSE
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)
```

---

get\_zenodo*get\_zenodo*

---

**Description**

get\_zenodo allows to get a Zenodo record, identified by its DOI or concept DOI.

**Usage**

```
get_zenodo(doi, sandbox = FALSE, logger = NULL)
```

**Arguments**

doi	a Zenodo DOI or concept DOI
sandbox	Use the sandbox infrastructure. Default is FALSE
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.

object of class `ZenodoRecord`

**Examples**

```
## Not run:  
get_zenodo("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](mailto:emmanuel.blondel1@gmail.com)>

**See Also**

Useful links:

- <https://github.com/eblondel/zen4R>
- Report bugs at <https://github.com/eblondel/zen4R/issues>

---

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

---

ZenodoException	<i>ZenodoException</i>
-----------------	------------------------

---

**Description**

ZenodoException

ZenodoException

**Format**

[R6Class](#) object.

**Value**

Object of [R6Class](#) for modelling a generic Zenodo service exception

**Super class**

[zen4R::zen4RLogger](#) -> ZenodoException

**Public fields**

message message

status status

**Methods****Public methods:**

- [ZenodoException\\$new\(\)](#)
- [ZenodoException\\$clone\(\)](#)

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

*Usage:*

`ZenodoException$new(message, status, logger = NULL)`

*Arguments:*

message message

status status

logger the logger type

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

*Usage:*

`ZenodoException$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>

---

ZenodoManager

*ZenodoManager*

---

**Description**

ZenodoManager

ZenodoManager

**Format**

[R6Class](#) object.

**Value**

Object of [R6Class](#) for modelling an ZenodoManager

**Super class**

[zen4R](#): : [zen4RLogger](#) -> ZenodoManager

**Public fields**

sandbox Zenodo manager sandbox status, TRUE if we interact with Sandbox infra

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

**Methods****Public methods:**

- [ZenodoManager\\$new\(\)](#)
- [ZenodoManager\\$getToken\(\)](#)
- [ZenodoManager\\$checkUserAuthentication\(\)](#)
- [ZenodoManager\\$getLanguages\(\)](#)
- [ZenodoManager\\$getLanguageById\(\)](#)
- [ZenodoManager\\$getLicenses\(\)](#)
- [ZenodoManager\\$getLicenseById\(\)](#)
- [ZenodoManager\\$getResourceTypes\(\)](#)
- [ZenodoManager\\$getResourceTypeById\(\)](#)
- [ZenodoManager\\$getCommunities\(\)](#)

- `ZenodoManager$getCommunityById()`
- `ZenodoManager$submitRecordToCommunities()`
- `ZenodoManager$removeRecordFromCommunities()`
- `ZenodoManager$getRecordCommunities()`
- `ZenodoManager$createReviewRequest()`
- `ZenodoManager$getReviewRequest()`
- `ZenodoManager$deleteReviewRequest()`
- `ZenodoManager$submitRecordForReview()`
- `ZenodoManager$getGrants()`
- `ZenodoManager$getAwards()`
- `ZenodoManager$getGrantsByName()`
- `ZenodoManager$getAwardsByName()`
- `ZenodoManager$getGrantById()`
- `ZenodoManager$getAwardById()`
- `ZenodoManager$getAffiliations()`
- `ZenodoManager$getAffiliationByName()`
- `ZenodoManager$getAffiliationById()`
- `ZenodoManager$getFunders()`
- `ZenodoManager$getFundersByName()`
- `ZenodoManager$getFunderById()`
- `ZenodoManager$getDepositions()`
- `ZenodoManager$getDepositionByConceptDOI()`
- `ZenodoManager$getDepositionByDOI()`
- `ZenodoManager$getDepositionById()`
- `ZenodoManager$getDepositionByConceptId()`
- `ZenodoManager$depositRecord()`
- `ZenodoManager$reserveDOI()`
- `ZenodoManager$deleteDOI()`
- `ZenodoManager$depositRecordVersion()`
- `ZenodoManager$deleteRecord()`
- `ZenodoManager$deleteRecordByDOI()`
- `ZenodoManager$deleteRecords()`
- `ZenodoManager$createEmptyRecord()`
- `ZenodoManager$editRecord()`
- `ZenodoManager$discardChanges()`
- `ZenodoManager$publishRecord()`
- `ZenodoManager$getFiles()`
- `ZenodoManager$getFile()`
- `ZenodoManager$startFileUpload()`
- `ZenodoManager$completeFileUpload()`
- `ZenodoManager$uploadFile()`
- `ZenodoManager$deleteFile()`

- `ZenodoManager$getRecords()`
- `ZenodoManager$getRecordByConceptDOI()`
- `ZenodoManager$getRecordByDOI()`
- `ZenodoManager$getRecordById()`
- `ZenodoManager$getRecordByConceptId()`
- `ZenodoManager$getRequests()`
- `ZenodoManager$getRequest()`
- `ZenodoManager$isActionableRequest()`
- `ZenodoManager$acceptRequest()`
- `ZenodoManager$declineRequest()`
- `ZenodoManager$cancelRequest()`
- `ZenodoManager$clone()`

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

*Usage:*

```
ZenodoManager$new(
  url = "https://zenodo.org/api",
  token = zenodo_pat(),
  sandbox = FALSE,
  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`

`sandbox` Indicates if the Zenodo sandbox platform should be used. Default is FALSE

`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** `checkUserAuthentication()`: Check whether the user is authenticated

*Usage:*

```
ZenodoManager$checkUserAuthentication()
```

**Method** `getLanguages()`: Get Languages supported by Zenodo.

*Usage:*

ZenodoManager\$getLanguages(pretty = TRUE)

*Arguments:*

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

*Returns:* list of languages as data.frame or list

**Method** getLanguageById(): Get language by Id.

*Usage:*

ZenodoManager\$getLanguageById(id)

*Arguments:*

id license id

*Returns:* the license

**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** getResourceTypes(): Get Resource types supported by Zenodo.

*Usage:*

ZenodoManager\$getResourceTypes(pretty = TRUE)

*Arguments:*

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

*Returns:* list of resource types as data.frame or list

**Method** getResourceTypeId(): Get resource type by Id.

*Usage:*

ZenodoManager\$getResourceTypeId(id)

*Arguments:*

id resource type id

*Returns:* the resource type

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

*Usage:*

```
ZenodoManager$getCommunities(pretty = TRUE, q = "", size = 500)
```

*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  
`q` an ElasticSearch compliant query, object of class character. Default is empty. Note that the Zenodo API restrains a maximum number of 10,000 records to be retrieved. Consequently, not all communities can be listed from Zenodo, a query has to be specified.  
`size` number of communities to be returned. By default equal to 500

*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** `submitRecordToCommunities()`: Submit a published record to one or more community

*Usage:*

```
ZenodoManager$submitRecordToCommunities(
  record,
  communities = list(),
  message = NULL
)
```

*Arguments:*

`record` an object of class [ZenodoRecord](#)

`communities` communities to which the record will be submitted

`message` message to send to the community curator(s), either a text or a named list for each community in case a community-specific message should be sent

*Returns:* a submission object of class `list`, or `NULL` if nothing was submitted

**Method** `removeRecordFromCommunities()`: Remove a record from one or more community

*Usage:*

```
ZenodoManager$removeRecordFromCommunities(record, communities = list())
```

*Arguments:*

`record` an object of class [ZenodoRecord](#)

`communities` communities to which the record will be submitted

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

**Method** `getRecordCommunities():` Get record communities

*Usage:*

`ZenodoManager$getRecordCommunities(record)`

*Arguments:*

`record` object of class `ZenodoRecord`

*Returns:* the list of communities in which the record was included

**Method** `createReviewRequest():` Creates a record review request in a community

*Usage:*

`ZenodoManager$createReviewRequest(record, community)`

*Arguments:*

`record` an object of class `ZenodoRecord`

`community` a community to which the record is submitted for review and publication

*Returns:* a review request object of class `list`, or `NULL` if nothing was submitted

**Method** `getReviewRequest():` Get a record review request

*Usage:*

`ZenodoManager$getReviewRequest(record)`

*Arguments:*

`record` an object of class `ZenodoRecord`

*Returns:* a review request object of class `list`, or `NULL` if nothing exists

**Method** `deleteReviewRequest():` Deletes a review request

*Usage:*

`ZenodoManager$deleteReviewRequest(record)`

*Arguments:*

`record` an object of class `ZenodoRecord`

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

**Method** `submitRecordForReview():` Submits a record for review. Prior to this submission, a community has to be selected for a record. This is done by using the method `createReviewRequest(record, community)`.

*Usage:*

`ZenodoManager$submitRecordForReview(recordId, message = NULL)`

*Arguments:*

`recordId` the ID of a `Zenodo` record

`message` message content for the submission. Optional

*Returns:* `TRUE` if submitted, `FALSE` otherwise

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

*Usage:*

`ZenodoManager$getGrants(q = "", pretty = TRUE, size = 500)`

*Arguments:*

*q* an Elasticsearch compliant query, object of class character. Default is empty. Note that the Zenodo API restrains a maximum number of 10,000 records to be retrieved. Consequently, not all grants can be listed from Zenodo, a query has to be specified.

*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 500.

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

**Method** `getAwards()`: Get Awards supported by Zenodo.

*Usage:*

```
ZenodoManager$getAwards(q = "", pretty = TRUE, size = 500)
```

*Arguments:*

*q* an Elasticsearch compliant query, object of class character. Default is empty. Note that the Zenodo API restrains a maximum number of 10,000 records to be retrieved. Consequently, not all awards can be listed from Zenodo, a query has to be specified.

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

*size* number of awards to be returned. By default equal to 500.

*Returns:* list of awards as data.frame or list

**Method** `getGrantsByName()`: Get grants by name. DEPRECATED: replaced by `getAwardByName`

*Usage:*

```
ZenodoManager$getGrantsByName(name, pretty = TRUE)
```

*Arguments:*

*name* name

*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

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

**Method** `getAwardsByName()`: Get awards by name.

*Usage:*

```
ZenodoManager$getAwardsByName(name, pretty = TRUE)
```

*Arguments:*

*name* name

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

*Returns:* list of awards as data.frame or list

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

*Usage:*

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

*Arguments:*

id grant id

*Returns:* the grant

**Method** `getAwardById()`: Get award by Id.

*Usage:*

`ZenodoManager$getAwardById(id)`

*Arguments:*

id award id

*Returns:* the award

**Method** `getAffiliations()`: Get Affiliations supported by Zenodo.

*Usage:*

`ZenodoManager$getAffiliations(q = "", pretty = TRUE, size = 500)`

*Arguments:*

q an ElasticSearch compliant query, object of class character. Default is empty. Note that the Zenodo API restrains a maximum number of 10,000 records to be retrieved. Consequently, not all affiliations can be listed from Zenodo, a query has to be specified.

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

size number of affiliations to be returned. By default equal to 500.

*Returns:* list of affiliations as data.frame or list

**Method** `getAffiliationByName()`: Get affiliations by name.

*Usage:*

`ZenodoManager$getAffiliationByName(name, pretty = TRUE)`

*Arguments:*

name name

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

*Returns:* list of affiliations as data.frame or list

**Method** `getAffiliationById()`: Get affiliation by Id.

*Usage:*

`ZenodoManager$getAffiliationById(id)`

*Arguments:*

id affiliation id

*Returns:* the affiliation

**Method** `getFunders()`: Get Funders supported by Zenodo based on a query.

*Usage:*

`ZenodoManager$getFunders(q = "", pretty = TRUE, size = 500)`

*Arguments:*

`q` an ElasticSearch compliant query, object of class `Character`. Default is empty. Note that the Zenodo API restrains a maximum number of 10,000 records to be retrieved. Consequently, not all funders can be listed from Zenodo, a query has to be specified.

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

`size` number of funders to be returned. By default equal to 500

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

**Method** `getFundersByName()`: Get funders by name.

*Usage:*

```
ZenodoManager$getFundersByName(name, pretty = TRUE)
```

*Arguments:*

`name` name

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

*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 (user records). By default 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 be 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, reservedDOI = TRUE, publish = FALSE)
```

*Arguments:*

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

`reservedDOI` reserve DOI. By default `TRUE`

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:* object of class ZenodoRecord

**Method** reserveDOI(): Reserves a DOI for a deposition (draft record)

*Usage:*

```
ZenodoManager$reserveDOI(record)
```

*Arguments:*

record the record to deposit, object of class ZenodoRecord

*Returns:* object of class ZenodoRecord

**Method** deleteDOI(): Reserves a DOI for a deposition (draft record)

*Usage:*

```
ZenodoManager$deleteDOI(record)
```

*Arguments:*

record the record for which DOI has to be deleted, object of class ZenodoRecord

*Returns:* object of class ZenodoRecord

**Method** depositRecordVersion(): Deposits a record version on Zenodo.

*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(reserveDOI = TRUE)

*Arguments:*

reserveDOI reserve DOI. By default TRUE

*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. Deleting a draft for an unpublished record will remove the draft and associated files from the system. Deleting a draft for a published record will remove the draft but not the published 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** getFile(): Get a file record metadata.

*Usage:*

ZenodoManager\$getFile(recordId, filename)

*Arguments:*

recordId the ID of the record.

filename filename

*Returns:* the file metadata

**Method** startFileUpload(): Start a file upload. The method will create a key for the file to be uploaded. This method is essentially for internal purpose, and is called directly in uploadFile for user convenience and for backward compatibility with the legacy Zenodo API.

*Usage:*

ZenodoManager\$startFileUpload(path, recordId)

*Arguments:*

path Local path of the file

recordId ID of the record

**Method** completeFileUpload(): Completes a file upload. The method will complete a file upload through a commit operation. This method is essentially for internal purpose, and is called directly in uploadFile for user convenience and for backward compatibility with the legacy Zenodo API.

*Usage:*

ZenodoManager\$completeFileUpload(path, recordId)

*Arguments:*

path Local path of the file

recordId ID of the record

**Method** uploadFile(): Uploads a file to a Zenodo record. With the new Zenodo Invenio RDM API, this method internally calls startFileUpload to create a file record (with a filename key) at start, followed by the actual file content upload. At this stage, the file upload is in "pending" status. At the end, the function calls completeFileUpload to commit the file which status becomes "completed".

*Usage:*

ZenodoManager\$uploadFile(path, record = NULL)

*Arguments:*

path Local path of the file  
record object of class ZenodoRecord

**Method deleteFile():** Deletes a file for a record. With the new Zenodo Invenio RDM API, if a file is deleted although its status was pending, only the upload content is deleted, and the file upload record (identified by a filename key) is kept. If the status was completed (with a file commit), the file record is deleted.

*Usage:*

ZenodoManager\$deleteFile(recordId, filename)

*Arguments:*

recordId ID of the record  
filename name of the file to be deleted

**Method getRecords():** Get the list of Zenodo records. By default 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 = TRUE)

*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 be 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** getRequests(): Search requests*Usage:*

ZenodoManager\$getRequests(q = "", sort = "bestmatch", size = 10)

*Arguments:*

q Search query used to filter results based on Elasticsearch's query string syntax. e.g. <https://www.elastic.co/guide/en/elastic-search/current/dsl-query-string-query.html#query-string-syntax>

sort Sort search results. Built-in options are "bestmatch", "name", "newest", "oldest" (default: "bestmatch" or "newest").

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

*Returns:* a list of ZenodoRecord

**Method** getRequest(): Get a request*Usage:*

ZenodoManager\$getRequest(request\_id)

*Arguments:*

request\_id the request ID

*Returns:* the request list object, NULL otherwise

**Method** isActionableRequest(): Checks if the request can be subject to an operation (accept, decline, cancel) depending on its status. To be subject to an operation, a request should not be closed or expired

*Usage:*

ZenodoManager\$isActionableRequest(request\_id)

*Arguments:*

request\_id the request ID

*Returns:* TRUE if

**Method** acceptRequest(): Accepts a request

*Usage:*

ZenodoManager\$acceptRequest(request\_id, message = NULL)

*Arguments:*

request\_id the request ID

message optional message reason for acceptance

*Returns:* TRUE if accepted, FALSE otherwise

**Method** declineRequest(): Declines a request

*Usage:*

ZenodoManager\$declineRequest(request\_id, message = NULL)

*Arguments:*

request\_id the request ID

message optional message reason for declination

*Returns:* TRUE if declined, FALSE otherwise

**Method** cancelRequest(): Cancels a request

*Usage:*

ZenodoManager\$cancelRequest(request\_id, message = NULL)

*Arguments:*

request\_id the request ID

message optional message reason for cancelation

*Returns:* TRUE if canceled, FALSE otherwise

**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

#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]]$filename)

## 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

created record creation date

updated record update date

revision\_id revision id

is\_draft is draft

is\_published is published

status record status

versions versions

access access policies

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

parent parent record

pids pids

stats stats

**Methods****Public methods:**

- ZenodoRecord\$new()
- ZenodoRecord\$getStats()
- ZenodoRecord\$getId()
- ZenodoRecord\$getParentId()
- ZenodoRecord\$getConceptId()
- ZenodoRecord\$setDOI()
- ZenodoRecord\$getDOI()
- ZenodoRecord\$getConceptDOI()
- ZenodoRecord\$setAccessPolicyRecord()
- ZenodoRecord\$setAccessPolicyFiles()
- ZenodoRecord\$setAccessPolicyEmbargo()
- ZenodoRecord\$setResourceType()
- ZenodoRecord\$setUploadType()
- ZenodoRecord\$setPublicationType()
- ZenodoRecord\$setImageType()
- ZenodoRecord\$setPublisher()
- ZenodoRecord\$setPublicationDate()
- ZenodoRecord\$addDate()
- ZenodoRecord\$removeDate()
- ZenodoRecord\$setTitle()
- ZenodoRecord\$addAdditionalTitle()
- ZenodoRecord\$removeAdditionalTitle()
- ZenodoRecord\$setDescription()
- ZenodoRecord\$addAdditionalDescription()
- ZenodoRecord\$removeAdditionalDescription()
- ZenodoRecord\$addPersonOrOrg()
- ZenodoRecord\$removePersonOrOrg()
- ZenodoRecord\$addCreator()
- ZenodoRecord\$removeCreatorByName()
- ZenodoRecord\$removeCreatorByAffiliation()
- ZenodoRecord\$removeCreatorByORCID()
- ZenodoRecord\$removeCreatorByGND()
- ZenodoRecord\$removeCreatorByISNI()
- ZenodoRecord\$removeCreatorByROR()
- ZenodoRecord\$addContributor()
- ZenodoRecord\$removeContributorByName()
- ZenodoRecord\$removeContributorByAffiliation()
- ZenodoRecord\$removeContributorByORCID()
- ZenodoRecord\$removeContributorByGND()
- ZenodoRecord\$removeContributorByISNI()

- `ZenodoRecord$removeContributorByROR()`
- `ZenodoRecord$addRight()`
- `ZenodoRecord$setLicense()`
- `ZenodoRecord$setVersion()`
- `ZenodoRecord$addLanguage()`
- `ZenodoRecord$setLanguage()`
- `ZenodoRecord$addRelatedIdentifier()`
- `ZenodoRecord$removeRelatedIdentifier()`
- `ZenodoRecord$setReferences()`
- `ZenodoRecord$addReference()`
- `ZenodoRecord$removeReference()`
- `ZenodoRecord$setSubjects()`
- `ZenodoRecord$setKeywords()`
- `ZenodoRecord$addSubject()`
- `ZenodoRecord$addKeyword()`
- `ZenodoRecord$removeSubject()`
- `ZenodoRecord$removeKeyword()`
- `ZenodoRecord$setNotes()`
- `ZenodoRecord$addFunding()`
- `ZenodoRecord$addGrant()`
- `ZenodoRecord$setGrants()`
- `ZenodoRecord$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\$addLocation()
- ZenodoRecord\$removeLocation()
- ZenodoRecord\$exportAs()
- ZenodoRecord\$exportAsBibTeX()
- ZenodoRecord\$exportAsCSL()
- ZenodoRecord\$exportAsDataCite()
- ZenodoRecord\$exportAsDataCiteXML()
- ZenodoRecord\$exportAsDataCiteJSON()
- ZenodoRecord\$exportAsDublinCore()
- ZenodoRecord\$exportAsDCAT()
- ZenodoRecord\$exportAsJSON()
- ZenodoRecord\$exportAsJSONLD()
- ZenodoRecord\$exportAsGeoJSON()
- ZenodoRecord\$exportAsMARCXML()
- ZenodoRecord\$exportAsCodeMeta()
- ZenodoRecord\$exportAsCFF()
- ZenodoRecord\$exportAsDataPackage()
- ZenodoRecord\$exportAsAllFormats()
- ZenodoRecord\$getCitation()
- ZenodoRecord\$listFiles()
- ZenodoRecord\$downloadFiles()
- ZenodoRecord\$print()
- ZenodoRecord\$toDCEntry()
- ZenodoRecord\$getFirstDOI()
- ZenodoRecord\$getLastDOI()
- ZenodoRecord\$getVersions()
- 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** `getStats()`: Get record statistics

*Usage:*

```
ZenodoRecord$getStats()
```

*Returns:* statistics as data.frame

**Method** getId(): Get the record Id

*Usage:*

```
ZenodoRecord$getId()
```

*Returns:* the Id, object of class character

**Method** getParentId(): Get the parent record Id

*Usage:*

```
ZenodoRecord$getParentId()
```

*Returns:* the parent Id, object of class character

**Method** getConceptId(): Get the concept record Id

*Usage:*

```
ZenodoRecord$getConceptId()
```

*Returns:* the concept Id, object of class character

**Method** setDOI(): Set the DOI. This method can be used if a DOI has been already assigned outside Zenodo.

*Usage:*

```
ZenodoRecord$setDOI(doi, provider = NULL, client = NULL)
```

*Arguments:*

doi DOI to set for the record

provider DOI provider

client DOI client

**Method** getDOI(): Get the record DOI.

*Usage:*

```
ZenodoRecord$getDOI()
```

*Returns:* the DOI, object of class character

**Method** getConceptDOI(): Get the concept (generic) DOI. The concept DOI is a generic DOI common to all versions of a Zenodo record.

*Usage:*

```
ZenodoRecord$getConceptDOI()
```

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

**Method** setAccessPolicyRecord(): Set the access policy for record, among values "public" (default) or "restricted" In Zenodo, in principle, the access policy 'restricted' is not available for records.

*Usage:*

```
ZenodoRecord$setAccessPolicyRecord(access = c("public", "restricted"))
```

*Arguments:*

access access policy ('public' or 'restricted')

**Method** `setAccessPolicyFiles()`: Set the access policy for files, among values "public" (default) or "restricted"

*Usage:*

```
ZenodoRecord$setAccessPolicyFiles(access = c("public", "restricted"))
```

*Arguments:*

access access policy ('public' or 'restricted')

**Method** `setAccessPolicyEmbargo()`: Set access policy embargo options

*Usage:*

```
ZenodoRecord$setAccessPolicyEmbargo(active = FALSE, until = NULL, reason = "")
```

*Arguments:*

active whether embargo is active or not. Default is FALSE

until embargo date, object of class Date. Default is NULL. Must be provided if embargo is active

reason embargo reason, object of class character. Default is an empty string

**Method** `setResourceType()`: Set the resource type (mandatory).

*Usage:*

```
ZenodoRecord$setResourceType(resourceType)
```

*Arguments:*

resourceType record resource type

**Method** `setUploadType()`: Set the upload type (mandatory). Deprecated since zen4R 1.0

*Usage:*

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

*Arguments:*

uploadType record upload type among the following values: 'publication', 'poster', 'presentation', 'dataset', 'image', 'video', 'software', 'lesson', 'physicalobject', 'other'

**Method** `setPublicationType()`: Set the publication type (mandatory if upload type is 'publication'). Deprecated since zen4R 1.0

*Usage:*

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

*Arguments:*

publicationType record publication type among the following values: 'annotationcollection', 'book', 'section', 'conferencepaper', 'datamanagementplan', 'article', 'patent', 'preprint', 'deliverable', 'milestone', 'proposal', 'report', 'softwaredocumentation', 'taxonomicreatment', 'technicalnote', 'thesis', 'workingpaper', 'other'

**Method** `setImageType()`: Set the image type (mandatory if image type is 'image'). Deprecated since zen4R 1.0

*Usage:*

ZenodoRecord\$setImageType(imageType)

*Arguments:*

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

**Method** setPublisher(): Set the publisher

*Usage:*

ZenodoRecord\$setPublisher(publisher)

*Arguments:*

publisher publisher object of class character

**Method** setPublicationDate(): Set the publication date. For more information on the accepted format, please check <https://inveniordm.docs.cern.ch/reference/metadata/#publication-date-1>

*Usage:*

ZenodoRecord\$setPublicationDate(publicationDate)

*Arguments:*

publicationDate object of class character

**Method** addDate(): Add date

*Usage:*

ZenodoRecord\$addDate(date, type, description = NULL)

*Arguments:*

date date

type type of date, among following values: 'accepted', 'available', 'collected', 'copyrighted', 'created', 'issued', 'other', 'submitted', 'updated', 'valid', 'withdrawn'

description free text, specific information about the date

**Method** removeDate(): Remove a date

*Usage:*

ZenodoRecord\$removeDate(date, type)

*Arguments:*

date the date to remove

type the date type of the date to be removed

*Returns:* TRUE if removed, FALSE otherwise

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

*Usage:*

ZenodoRecord\$setTitle(title)

*Arguments:*

title object of class character

**Method** addAdditionalTitle(): Add additional record title

*Usage:*

```
ZenodoRecord$addAdditionalTitle(title, type, lang = "eng")
```

*Arguments:*

title title free text

type type of title, among following values: alternative-title, subtitle, translated-title, other

lang language id

*Returns:* TRUE if added, FALSE otherwise

**Method** removeAdditionalTitle(): Removes additional record title.

*Usage:*

```
ZenodoRecord$removeAdditionalTitle(title, type, lang = "eng")
```

*Arguments:*

title title free text

type type of title, among following values: abstract, methods, series-information, table-of-contents, technical-info, other

lang language id

*Returns:* TRUE if removed, FALSE otherwise

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

*Usage:*

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

*Arguments:*

description object of class character

**Method** addAdditionalDescription(): Add additional record description

*Usage:*

```
ZenodoRecord$addAdditionalDescription(description, type, lang = "eng")
```

*Arguments:*

description description free text

type type of description, among following values: abstract, methods, series-information, table-of-contents, technical-info, other

lang language id

*Returns:* TRUE if added, FALSE otherwise

**Method** removeAdditionalDescription(): Removes additional record description

*Usage:*

```
ZenodoRecord$removeAdditionalDescription(description, type, lang = "eng")
```

*Arguments:*

description description free text

type type of description, among following values: abstract, methods, series-information, table-of-contents, technical-info, other

lang language id

*Returns:* TRUE if removed, FALSE otherwise

**Method** `addPersonOrOrg()`: Add a person or organization for the record. For persons, the approach is to use the `firstname` and `lastname` arguments, that by default will be concatenated for Zenodo as `lastname, firstname`. For organizations, use the `name` argument.

*Usage:*

```
ZenodoRecord$addPersonOrOrg(
  firstname = NULL,
  lastname = NULL,
  name = paste(lastname, firstname, sep = ", "),
  orcid = NULL,
  gnd = NULL,
  isni = NULL,
  ror = NULL,
  role = NULL,
  affiliations = NULL,
  sandbox = FALSE,
  type
)
```

*Arguments:*

`firstname` person first name

`lastname` person last name

`name` organization name

`orcid` person or organization ORCID (optional)

`gnd` person or organization GND (optional)

`isni` person or organization ISNI (optional)

`ror` person or organization ROR (optional)

`role` role, values among: `contactperson`, `datacollector`, `datacurator`, `datamanager`, `distributor`, `editor`, `funder`, `hostinginstitution`, `producer`, `projectleader`, `projectmanager`, `projectmember`, `registrationagency`, `registrationauthority`, `relatedperson`, `researcher`, `researchgroup`, `rightsholder`, `supervisor`, `sponsor`, `workpackageleader`, `other`

`affiliations` person or organization affiliations (optional)

`sandbox` Use the Zenodo sandbox infrastructure as basis to control available affiliations. Default is FALSE

`type` type of person or org (creators/contributors)

*Returns:* TRUE if added, FALSE otherwise

**Method** `removePersonOrOrg()`: Removes a person or organization by a property. The `by` parameter should be the name of the person or organization property (`'name'`, `'affiliation'`, `'orcid'`, `'gnd'`, `'isni'`, `'ror'`).

*Usage:*

```
ZenodoRecord$removePersonOrOrg(by, property, type)
```

*Arguments:*

`by` property used as criterion to remove the person or organization

`property` property value used to remove the person or organization

`type` type of person or org (creators / contributors)

*Returns:* TRUE if removed, FALSE otherwise

**Method** `addCreator()`: Add a creator for the record. For persons, the approach is to use the `firstname` and `lastname` arguments, that by default will be concatenated for Zenodo as `lastname, firstname`. For organizations, use the `name` argument.

*Usage:*

```
ZenodoRecord$addCreator(
  firstname = NULL,
  lastname = NULL,
  name = paste(lastname, firstname, sep = ", "),
  orcid = NULL,
  gnd = NULL,
  isni = NULL,
  ror = NULL,
  role = NULL,
  affiliations = NULL,
  sandbox = FALSE
)
```

*Arguments:*

`firstname` person first name

`lastname` person last name

`name` organization name

`orcid` creator ORCID (optional)

`gnd` creator GND (optional)

`isni` creator ISNI (optional)

`ror` creator ROR (optional)

`role` role, values among: `contactperson`, `datacollector`, `datacurator`, `datamanager`, `distributor`, `editor`, `funder`, `hostinginstitution`, `producer`, `projectleader`, `projectmanager`, `projectmember`, `registrationagency`, `registrationauthority`, `relatedperson`, `researcher`, `researchgroup`, `rightsholder`, `supervisor`, `sponsor`, `workpackageleader`, `other`

`affiliations` creator affiliations (optional)

`sandbox` Use the Zenodo sandbox infrastructure as basis to control available affiliations. Default is FALSE

*Returns:* TRUE if added, 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** removeCreatorByISNI(): Removes a creator by ISNI.

*Usage:*

ZenodoRecord\$removeCreatorByISNI(isni)

*Arguments:*

isni creator ISNI

*Returns:* TRUE if removed, FALSE otherwise

**Method** removeCreatorByROR(): Removes a creator by ROR.

*Usage:*

ZenodoRecord\$removeCreatorByROR(ror)

*Arguments:*

ror creator ROR

*Returns:* TRUE if removed, FALSE otherwise

**Method** addContributor(): Add a contributor for the record. For persons, the approach is to use the `firstname` and `lastname` arguments, that by default will be concatenated for Zenodo as `lastname, firstname`. For organizations, use the `name` argument.

*Usage:*

```
ZenodoRecord$addContributor(  
  firstname = NULL,  
  lastname = NULL,  
  name = paste(lastname, firstname, sep = ", "),  
  orcid = NULL,
```

```

    gnd = NULL,
    isni = NULL,
    ror = NULL,
    role = NULL,
    affiliations = NULL,
    sandbox = FALSE
)

```

*Arguments:*

firstname person first name

lastname person last name

name organization name

orcid contributor ORCID (optional)

gnd contributor GND (optional)

isni contributor ISNI (optional)

ror contributor ROR (optional)

role role, values among: contactperson, datacollector, datacurator, datamanager, distributor, editor, funder, hostinginstitution, producer, projectleader, projectmanager, projectmember, registrationagency, registrationauthority, relatedperson, researcher, researchgroup, rightsholder, supervisor, sponsor, workpackageleader, other

affiliations contributor affiliations (optional)

sandbox Use the Zenodo sandbox infrastructure as basis to control available affiliations. Default is FALSE

*Returns:* TRUE if added, 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** removeContributorByISNI(): Removes a contributor by ISNI.

*Usage:*

```
ZenodoRecord$removeContributorByISNI(isni)
```

*Arguments:*

isni contributor ISNI

*Returns:* TRUE if removed, FALSE otherwise

**Method** removeContributorByROR(): Removes a contributor by ROR.

*Usage:*

```
ZenodoRecord$removeContributorByROR(ror)
```

*Arguments:*

ror contributor ROR

*Returns:* TRUE if removed, FALSE otherwise

**Method** addRight(): Add right/license. Please see <https://inveniordm.docs.cern.ch/reference/metadata/#rights-licenses-0-n>

*Usage:*

```
ZenodoRecord$addRight(
  id = NULL,
  title = NULL,
  description = NULL,
  link = NULL,
  sandbox = FALSE
)
```

*Arguments:*

id license id

title license title

description a multi-lingual list

link license link

sandbox Use the Zenodo sandbox infrastructure as basis to control available licenses. Default is FALSE

**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 be fetched with the ZenodoManager and the function \$getLicenses().

*Usage:*

```
ZenodoRecord$setLicense(licenseId, sandbox = FALSE)
```

*Arguments:*

licenseId a license Id

sandbox Use the Zenodo sandbox infrastructure as basis to control available licenses. Default is FALSE

*Returns:* TRUE if set, FALSE otherwise

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

*Usage:*

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

*Arguments:*

version the record version to set

**Method** addLanguage(): Adds a language.

*Usage:*

```
ZenodoRecord$addLanguage(language)
```

*Arguments:*

language ISO 639-2 or 639-3 code

**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 scheme and relation type.

*Usage:*

```
ZenodoRecord$addRelatedIdentifier(
  identifier,
  scheme,
  relation_type,
  resource_type = NULL
)
```

*Arguments:*

identifier identifier

scheme scheme among following values: ark, arxiv, bibcode, doi, ean13, eissn, handle, igsn, isbn, issn, istic, lissn, lsid, pubmed id, purl, upc, url, urn, w3id

relation\_type relation type among following values: iscitedby, cites, issupplementto, is-supplementedby, iscontinuedby, continues, isdescribedby, describes, hasmetadata, ismeta-datafor, isnewversionof, ispreviousversionof, ispartof, haspart, isreferencedby, references, isdocumentedby, documents, iscompiledby, compiles, isvariantformof, isoriginalformof, isidenticalto, isalternateidentifier, isreviewedby, reviews, isderivedfrom, issourceof, requires, isrequiredby, isobsoletedby, obsoletes

resource\_type optional resource type

*Returns:* TRUE if added, FALSE otherwise

**Method** removeRelatedIdentifier(): Removes a related identifier with a given scheme/relation\_type

*Usage:*

ZenodoRecord\$removeRelatedIdentifier(identifier, scheme, relation\_type)

*Arguments:*

identifier identifier

scheme scheme among following values: ark, arxiv, bibcode, doi, ean13, eissn, handle, igsn, isbn, issn, istc, lissn, lsid, pubmed id, purl, upc, url, urn, w3id

relation\_type relation type among following values: iscitedby, cites, issupplementto, is-supplementedby, iscontinuedby, continues, isdescribedby, describes, hasmetadata, ismeta-datafor, isnewversionof, ispreviousversionof, ispartof, haspart, isreferencedby, references, isdocumentedby, documents, iscompiledby, compiles, isvariantformof, isoriginalformof, isidenticalto, isalternateidentifier, isreviewedby, reviews, isderivedfrom, issourceof, requires, isrequiredby, isobsoletedby, obsoletes

*Returns:* TRUE if removed, FALSE otherwise

**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** setSubjects(): Set subjects

*Usage:*

ZenodoRecord\$setSubjects(subjects)

*Arguments:*

subjects a vector or list of subjects to set for the record

**Method** setKeywords(): Set keywords

*Usage:*

ZenodoRecord\$setKeywords(keywords)

*Arguments:*

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

**Method** addSubject(): Add a subject

*Usage:*

ZenodoRecord\$addSubject(subject)

*Arguments:*

subject the subject to add

*Returns:* TRUE if added, FALSE otherwise

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

*Usage:*

ZenodoRecord\$addKeyword(keyword)

*Arguments:*

keyword the keyword to add

*Returns:* TRUE if added, FALSE otherwise

**Method** removeSubject(): Remove a subject

*Usage:*

ZenodoRecord\$removeSubject(subject)

*Arguments:*

subject the subject to remove

*Returns:* TRUE if removed, FALSE otherwise

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

*Usage:*

ZenodoRecord\$removeKeyword(keyword)

*Arguments:*

keyword the keyword to remove

*Returns:* TRUE if removed, FALSE otherwise

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

*Usage:*

ZenodoRecord\$setNotes(notes)

*Arguments:*

notes object of class character

**Method** addFunding(): Adds funding. Used internally, prefer using addGrant instead.

*Usage:*

```
ZenodoRecord$addFunding(funder = NULL, grant = NULL, sandbox = FALSE)
```

*Arguments:*

funder funder id or name

grant grant id or title

sandbox Use the Zenodo sandbox infrastructure as basis to control available grants. Default is FALSE

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

*Usage:*

```
ZenodoRecord$addGrant(grant, sandbox = FALSE)
```

*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 \$getAwards().

sandbox Use the Zenodo sandbox infrastructure as basis to control available grants. Default is FALSE

*Returns:* TRUE if added, FALSE otherwise

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

*Usage:*

```
ZenodoRecord$setGrants(grants, sandbox = FALSE)
```

*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 \$getAwards(). Each grant should be set with the Zenodo id of the grant If not recognized by Zenodo, the function will raise a warning only.

sandbox Use the Zenodo sandbox infrastructure as basis to control available grants. Default is FALSE

**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** `addLocation()`: Adds a location to the record metadata.

*Usage:*

```
ZenodoRecord$addLocation(place, description = NULL, lat = NULL, lon = NULL)
```

*Arguments:*

place place (required)

description description

lat latitude

lon longitude

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

*Usage:*

```
ZenodoRecord$removeLocation(place)
```

*Arguments:*

place place (required)

*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* a valid Zenodo export format among the following: BibTeX, CSL, DataCite (or DataCiteXML), DublinCore, DCAT, JSON, JSON-LD, GeoJSON, MARCXML, DataCiteJSON, CodeMeta, DataPackage, CFF.

*filename* the target filename (without extension)

*append\_format* whether 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 written file name (with extension)

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

*Usage:*

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

*Arguments:*

*filename* the target filename (without extension)

*Returns:* the written file name (with extension)

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

*Usage:*

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

*Arguments:*

*filename* the target filename (without extension)

*Returns:* the written file name (with extension)

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

*Usage:*

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

*Arguments:*

*filename* the target filename (without extension)

*Returns:* the written file name (with extension)

**Method** `exportAsDataCiteXML()`: Exports record as DataCite XML

*Usage:*

```
ZenodoRecord$exportAsDataCiteXML(filename)
```

*Arguments:*

*filename* the target filename (without extension)

*Returns:* the written file name (with extension)

**Method** `exportAsDataCiteJSON()`: Exports record as DataCite JSON

*Usage:*

```
ZenodoRecord$exportAsDataCiteJSON(filename)
```

*Arguments:*

filename the target filename (without extension)

*Returns:* the written file name (with extension)

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

*Usage:*

ZenodoRecord\$exportAsDublinCore(filename)

*Arguments:*

filename the target filename (without extension)

*Returns:* the written file name (with extension)

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

*Usage:*

ZenodoRecord\$exportAsDCAT(filename)

*Arguments:*

filename the target filename (without extension)

*Returns:* the written file name (with extension)

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

*Usage:*

ZenodoRecord\$exportAsJSON(filename)

*Arguments:*

filename the target filename (without extension)

*Returns:* the written 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 written file name (with extension)

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

*Usage:*

ZenodoRecord\$exportAsMARCXML(filename)

*Arguments:*

filename the target filename (without extension)

*Returns:* the written file name (with extension)

**Method** exportAsCodeMeta(): Exports record as CodeMeta

*Usage:*

```
ZenodoRecord$exportAsCodeMeta(filename)
```

*Arguments:*

filename the target filename (without extension)

*Returns:* the written file name (with extension)

**Method** exportAsCFF(): Exports record as Citation File Format

*Usage:*

```
ZenodoRecord$exportAsCFF(filename)
```

*Arguments:*

filename the target filename (without extension)

*Returns:* the written file name (with extension)

**Method** exportAsDataPackage(): Exports record as DataPackage

*Usage:*

```
ZenodoRecord$exportAsDataPackage(filename)
```

*Arguments:*

filename the target filename (without extension)

*Returns:* the written 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** getCitation(): Get record citation

*Usage:*

```
ZenodoRecord$getCitation(  
  style = c("havard-cite-them-right", "apa", "modern-language-association", "vancouver",  
           "chicago-fullnote-bibliography", "ieee", "bibtex")  
)
```

*Arguments:*

style the style character string among. Possible values "havard-cite-them-right", "apa", "modern-language-association", "vancouver", "chicago-fullnote-bibliography", "ieee", or "bibtex"

*Returns:* the citation text

**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,
  overwrite = TRUE,
  timeout = 60,
  ...
)
```

*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).

`overwrite` (default is TRUE) can be set to FALSE to avoid re-downloading existing files.

`timeout` (default is 60s) see `download.file`.

... 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). Attempts 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** `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** `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

Internal method. Prefer using `addCreator` or `addContributor`

Internal method. Prefer using `removeCreator` or `removeContributor`

See examples in [download\\_zenodo](#) utility function.

## Author(s)

Emmanuel Blondel <emmanuel.blondel1@gmail.com>

---

ZenodoRequest	<i>ZenodoRequest</i>
---------------	----------------------

---

**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\\$hasException\(\)](#)
- [ZenodoRequest\\$getResult\(\)](#)
- [ZenodoRequest\\$setResult\(\)](#)
- [ZenodoRequest\\$clone\(\)](#)

**Method** [new\(\)](#): Initializes a ZenodoRequest

*Usage:*

```
ZenodoRequest$new(  
  url,  
  type,  
  request,  
  data = NULL,  
  file = NULL,  
  progress = FALSE,  
  accept = "application/vnd.inveniordm.v1+json",
```

```
    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)

`progress` whether a progress status has to be displayed for download/upload

`accept` accept header. Default is "application/vnd.inveniordm.v1+json"

`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** `hasException()`: has exception?

*Usage:*

```
ZenodoRequest$hasException()
```

*Returns:* TRUE if request has exception, FALSE otherwise

**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

- \* **Exception**
  - ZenodoException, [10](#)
- \* **Request**
  - ZenodoRequest, [54](#)
- \* **Zenodo**
  - ZenodoException, [10](#)
  - ZenodoRequest, [54](#)
- \* **logger**
  - zen4RLogger, [8](#)
- \* **manager**
  - ZenodoManager, [11](#)
- \* **record**
  - ZenodoRecord, [28](#)
- \* **zenodo**
  - ZenodoManager, [11](#)
  - ZenodoRecord, [28](#)

DCEntry, [53](#)  
download\_zenodo, [2](#), [53](#)  
  
export\_zenodo, [3](#)  
  
get\_citation, [4](#)  
get\_licenses, [5](#)  
get\_versions, [6](#)  
get\_zenodo, [6](#)  
  
R6Class, [8](#), [10](#), [11](#), [28](#), [54](#)  
  
zen4R, [7](#)  
zen4R-package (zen4R), [7](#)  
zen4R::zen4RLogger, [10](#), [11](#), [28](#), [54](#)  
zen4RLogger, [8](#)  
zenodo\_pat, [13](#), [56](#)  
ZenodoException, [10](#)  
ZenodoManager, [11](#)  
ZenodoRecord, [3](#), [15](#), [16](#), [28](#), [31](#), [52](#)  
ZenodoRequest, [54](#)