Package 'AnnotationGx'

November 25, 2026

Title AnnotationGx: A package for building, updating and querying an annotation database for pharmaco-genomic data

Version 1.0.0

Description A package for building, updating and querying an annotation database for pharmaco-genomic data. Provides API interfaces to various bioinformatic and chemoinformatic databases, as well as a set of functions for curating your data using the annotations retrieved from those APIs.

Depends R (>= 4.4.0)

Imports BiocFileCache, checkmate, cli, crayon, data.table, httr2, memoise, parallel, readr, R6, methods, xml2

Suggests BiocStyle, testthat (>= 3.2.0), covr, knitr, rmarkdown, RefManageR, sessioninfo

BugReports https://github.com/bhklab/AnnotationGx/issues/new?template=issue_template.md

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first watcher, parallel*

License GPL-3

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Encoding UTF-8

URL https://bhklab.github.io/AnnotationGx/

biocViews Annotation, Pharmacogenomics, Cheminformatics

Config/roxygen2/version 8.0.0

git_url https://git.bioconductor.org/packages/AnnotationGx

git_branch RELEASE_3_24

git_last_commit 75e0d2b

git_last_commit_date 2026-11-23

Repository Bioconductor 3.24

Date/Publication 2026-11-24

Author Michael Tran [aut],
Jermiah Joseph [aut],
Christopher Eeles [aut],
Benjamin Haibe-Kains [aut, cre]

Maintainer Benjamin Haibe-Kains <benjamin.haibe.kains@utoronto.ca>

Contents

AnnotationGx-package . . . 3
annotateCellAccession . . . 3
annotatePubchemCompound . . . 4
AttributeInfo . . . 5
AttributeSet . . . 6
BioMartClient . . . 7
bm_query_builder . . . 9
CCLE_sampleMetadata . . . 10
CCLE_treatmentMetadata . . . 11
cellosaurusAPIVersion . . . 11
cellosaurus_fields . . . 12
cleanCharacterStrings . . . 12
CTRP_sampleMetadata . . . 13
CTRP_treatmentMetadata . . . 13
DatasetInfo . . . 14
FilterInfo . . . 15
gCSI_sampleMetadata . . . 17
gCSI_treatmentMetadata . . . 17
getChemblFilterTypes . . . 18
getChemblMechanism . . . 18
getChemblResourceFields . . . 19
getChemblResources . . . 20
getOncotreeMainTypes . . . 20
getOncotreeTumorTypes . . . 21
getOncotreeVersions . . . 21
getPubchemAnnotationHeadings . . . 22
getPubchemCompound . . . 22
getPubchemProperties . . . 23
getPubchemStatus . . . 24
getUnichemSources . . . 24
mapCell2Accession . . . 25
mapCID2Properties . . . 26
mapCompound2CID . . . 27
MartInfo . . . 28
matchNested . . . 29
queryChemblAPI . . . 30
queryUnichemCompound . . . 31
query_hgnc_by_genes . . . 32
standardize_names . . . 33
strSplit . . . 33
unlistNested . . . 34

Index **35**

`AnnotationGx-package` *AnnotationGx: AnnotationGx: A package for building, updating and querying an annotation database for pharmaco-genomic data*

Description

A package for building, updating and querying an annotation database for pharmaco-genomic data. Provides API interfaces to various bioinformatic and chemoinformatic databases, as well as a set of functions for curating your data using the annotations retrieved from those APIs.

Author(s)

Maintainer: Benjamin Haibe-Kains <benjamin.haibe.kains@utoronto.ca>

Authors:

- Benjamin Haibe-Kains <benjamin.haibe.kains@utoronto.ca>
- Michael Tran <michaelcao-anh.tran@uhn.ca>
- Jermiah Joseph <jermiah.joseph@gmail.com>
- Christopher Eeles <christopher.eeles@uhnresearch.ca>

See Also

Useful links:

- https://bhklab.github.io/AnnotationGx/
- Report bugs at https://github.com/bhklab/AnnotationGx/issues/new?template=issue_template.md

---

`annotateCellAccession` *Annotate Cell Accession*

Description

This function takes a Cellosaurus accession and returns annotations for the cell line.

Usage

```
annotateCellAccession(
  accessions,
  to = c("id", "ac", "hi", "sy", "ca", "sx", "ag", "di", "derived-from-site",
    "misspelling", "dt"),
  query_only = FALSE,
  raw = FALSE
)
```

Arguments

accessions	The Cellosaurus accession to annotate.
to	A character vector specifying the types of annotations to retrieve. Possible values include "id", "ac", "hi", "sy", "ca", "sx", "ag", "di", "derived-from-site", "misspelling", and "dt".
query_only	A logical value indicating whether to only return the query string.
raw	A logical value indicating whether to return the raw response.

Value

A data frame containing the annotations for the cell line.

Examples

```
annotateCellAccession("CVCL_0031")
annotateCellAccession("CVCL_0031", to = c("id", "ac", "hi", "sy"))
```

---

annotatePubchemCompound

Annotate PubChem Compound

Description

This function retrieves information about a PubChem compound based on the provided compound ID (CID).

Usage

```
annotatePubchemCompound(
  cids,
  heading = "ChEMBL ID",
  source = NULL,
  parse_function = identity,
  query_only = FALSE,
  raw = FALSE,
  nParallel = 1
)
```

Arguments

cids	The compound ID (CID) of the PubChem compound.
heading	The type of information to retrieve. Default is "ChEMBL ID".
source	The data source to use. Default is NULL.
parse_function	A custom parsing function to process the response. Default is the identity function.
query_only	Logical indicating whether to return the query URL only. Default is FALSE.
raw	Logical indicating whether to return the raw response. Default is FALSE.
nParallel	The number of parallel processes to use. Default is 1.

Value

The annotated information about the PubChem compound.

Examples

```
annotatePubchemCompound(cid = 2244)
annotatePubchemCompound(cid = c(2244, 67890), heading = "CAS")
```

`AttributeInfo` *BioMart Attribute Information Class*

Description

Represents an attribute from a BioMart dataset.

Details

This class encapsulates information about an attribute available in a BioMart dataset. Attributes are data fields that can be selected for retrieval in BioMart query results.

Public fields

`name` The internal name of the attribute used in BioMart queries

`displayName` The human-readable name of the attribute shown in user interfaces

`description` A textual description explaining what the attribute represents

`linkURL` URL that provides additional information about the attribute

`isHidden` Logical flag indicating if the attribute should be hidden in user interfaces

Methods

Public methods:

- `AttributeInfo$new()`
- `AttributeInfo$print()`
- `AttributeInfo$clone()`

`AttributeInfo$new()`: Initialize a new AttributeInfo object

Usage:

```
AttributeInfo$new(
  name,
  displayName = NULL,
  description = NULL,
  linkURL = NULL,
  isHidden = NULL
)
```

Arguments:

`name` Character, the internal name of the attribute used in API calls

`displayName` Character, the human-readable name shown to users

`description` Character, the attribute description explaining what it represents
`linkURL` Character, URL for additional information about the attribute
`isHidden` Logical, whether the attribute should be hidden in UIs

Returns: A new AttributeInfo object

`AttributeInfo$print()`: Print method for AttributeInfo objects

Usage:

```
AttributeInfo$print(...)
```

Arguments:

`...` Additional arguments passed to print methods (not used)

Returns: Invisibly returns self for method chaining

`AttributeInfo$clone()`: The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`AttributeSet` *BioMart Attribute Set Class*

Description

Represents a set of attributes from a BioMart dataset.

Public fields

`attributes` List of AttributeInfo objects

Methods

Public methods:

- `AttributeSet$new()`
- `AttributeSet$get_by_display_name()`
- `AttributeSet$print()`
- `AttributeSet$as.list()`
- `AttributeSet$filter()`
- `AttributeSet$clone()`

`AttributeSet$new()`: Initialize a new AttributeSet object

Usage:

```
AttributeSet$new(attributes)
```

Arguments:

`attributes` List of AttributeInfo objects

`AttributeSet$get_by_display_name()`:

Usage:

```
AttributeSet$get_by_display_name(display_names)
```

Arguments:

`display_names` Character vector of display names

Returns: List of matching AttributeInfo objects

`AttributeSet$print()`**:** Print method for AttributeSet objects

Usage:

```
AttributeSet$print(...)
```

Arguments:

`...` Additional arguments passed to print methods (not used)

Returns: Invisibly returns self

`AttributeSet$as.list()`**:** Convert AttributeSet to list

Usage:

```
AttributeSet$as.list()
```

Returns: List of attributes DisplayName as strings

`AttributeSet$filter()`**:** Filter attributes based on regex pattern

Usage:

```
AttributeSet$filter(pattern, exclude = FALSE)
```

Arguments:

`pattern` Regular expression pattern to match against display names
`exclude` Logical, if TRUE excludes matching patterns, if FALSE includes them

Returns: A new AttributeSet with filtered attributes

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`BioMartClient` *BioMart API Client*

Description

Client class for interacting with the BioMart REST API.

Details

This class provides methods for querying BioMart REST API endpoints, retrieving information about available marts, datasets, attributes, and filters.

Public fields

`base_url` Base URL of the BioMart service

`path` Path to the BioMart API on the server

Methods

Public methods:

- `BioMartClient$new()`
- `BioMartClient$get_marts()`
- `BioMartClient$get_datasets()`
- `BioMartClient$get_attributes()`
- `BioMartClient$get_filters()`
- `BioMartClient$clone()`

`BioMartClient$new()`: Initialize a new BioMartClient

Usage:

```
BioMartClient$new(base_url, path = "/biomart")
```

Arguments:

`base_url` Character, the base URL of the BioMart service

`path` Character, the API path, defaults to "/biomart"

Returns: A new BioMartClient object

`BioMartClient$get_marts()`: Retrieve available marts from the BioMart service

Usage:

```
BioMartClient$get_marts()
```

Returns: List of MartInfo objects

`BioMartClient$get_datasets()`: Retrieve available datasets for a given mart

Usage:

```
BioMartClient$get_datasets(mart)
```

Arguments:

`mart` MartInfo object, the mart to query

Returns: List of DatasetInfo objects

`BioMartClient$get_attributes()`: Retrieve available attributes for a given dataset

Usage:

```
BioMartClient$get_attributes(dataset)
```

Arguments:

`dataset` DatasetInfo object, the dataset to query

Returns: List of attribute information

`BioMartClient$get_filters()`: Retrieve available filters for a given dataset

Usage:

```
BioMartClient$get_filters(dataset)
```

Arguments:

`dataset` DatasetInfo object, the dataset to query

Returns: List of filter information

`BioMartClient$clone()`: The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create a client object (no network call required)
client <- BioMartClient$new("https://biomart.genenames.org")
client$path


# Create a client for HGNC BioMart
client <- BioMartClient$new("https://biomart.genenames.org")

# Get available marts
marts <- client$get_marts()

# Select a mart and get its datasets
hgnc <- marts[[1]]
datasets <- client$get_datasets(hgnc)
```

`bm_query_builder` *Build BioMart Query XML*

Description

Build BioMart Query XML

Usage

```
bm_query_builder(
  dataset,
  filters = list(),
  attributes = character(),
  client_name = "biomartclient",
  processor = "TSV",
  header = TRUE,
  limit = -1
)
```

Arguments

dataset	A DatasetInfo object representing the BioMart dataset to query
filters	List of filters, either: • Named list where names are filter names and values are filter values • List of FilterInfo objects with added 'value' field
attributes	AttributeSet, list of AttributeInfo objects, or character vector specifying the attributes to retrieve
client_name	Character, name of client for query identification (default: "biomartclient")
processor	Character, output format processor type (default: "TSV")
header	Logical, whether to include header row in results (default: TRUE)
limit	Integer, maximum number of rows to return, -1 for unlimited (default: -1)

Value

Character string containing the formatted XML BioMart query

CCLE_sampleMetadata *CCLE Sample Metadata*

Description

A column-reduced example sample metadata table derived from Cancer Cell Line Encyclopedia/DepMap resources.

Usage

```
data(CCLE_sampleMetadata)
```

Format

A `data.table` with 1461 rows and 4 columns:

CCLE_ID CCLE cell line identifier.

depMapID DepMap identifier.

Name Cell line display name.

CCLE_ID_parsed Parsed cell line identifier used by examples.

Source

CCLE/DepMap resources; BHKLab-curated PSet on ORCESTRA, https://www.orcestra.ca/pset/10.5281/zenodo.3905462. See `system.file("extdata", "LICENSES.md", package = "AnnotationGx")` for bundled data provenance notes.

Examples

```
data(CCLE_sampleMetadata)
head(CCLE_sampleMetadata)
```

CCLE_treatmentMetadata *CCLE Treatment Metadata*

Description

A column-reduced example treatment metadata table derived from legacy CCLE pharmacological profiling data.

Usage

```
data(CCLE_treatmentMetadata)
```

Format

A `data.table` with 24 rows and 1 column:

CCLE.treatmentid CCLE treatment name.

Source

CCLE legacy pharmacological profiling data, https://data.broadinstitute.org/ccle_legacy_data/pharmacological_profiling/CCLE_NP24.2009_profiling_2012.02.20.csv; BHKLab-curated PSet on ORCESTRA, https://www.orcestra.ca/pset/10.5281/zenodo.3905462. See `system.file("extdata", "LICENSES.md", package = "AnnotationGx")` for bundled data provenance notes.

Examples

```
data(CCLE_treatmentMetadata)
head(CCLE_treatmentMetadata)
```

cellosaurusAPIVersion *Get the Cellosaurus API version*

Description

This function retrieves the version of the Cellosaurus API.

Usage

```
cellosaurusAPIVersion()
```

Value

The version of the Cellosaurus API.

Examples

```
cellosaurusAPIVersion()
```

cellosaurus_fields *Get the list of fields in the Cellosaurus schema*

Description

This function retrieves the list of fields available in the Cellosaurus schema. It internally calls the `.cellosaurus_schema()` function to fetch the schema and extracts the list of fields from it.

Usage

```
cellosaurus_fields(common = FALSE, upper = FALSE)
```

Arguments

common	Logical indicating whether to return only the common fields. Default is FALSE.
upper	Logical indicating whether to return the fields in uppercase. Default is FALSE.

Value

A character vector containing the list of fields in the Cellosaurus schema.

Examples

```
cellosaurus_fields()
cellosaurus_fields(common = TRUE)
cellosaurus_fields(upper = TRUE)
```

cleanCharacterStrings *Clean character strings by removing special characters and formatting.*

Description

This function takes a character string as input and performs several cleaning operations to remove special characters, formatting, and unwanted substrings. The cleaned string is then returned as the output.

Usage

```
cleanCharacterStrings(name, space_action = "")
```

Arguments

name	A character string to be cleaned.
space_action	A character vector specifying the actions to be taken for space characters. One of c("", "-", " ").

Value

The cleaned character string.

Examples

```
cleanCharacterStrings("Cisplatin: 1 mg/mL (1.5 mM); 5 mM in DMSO")
```

CTRP_sampleMetadata *CTRP Sample Metadata*

Description

A column-reduced example sample metadata table derived from Cancer Therapeutics Response Portal v2 metadata.

Usage

```
data(CTRP_sampleMetadata)
```

Format

A `data.table` with 1107 rows and 2 columns:

master_ccl_id CTRP master cell line identifier.

ccl_name CTRP cell line name.

Source

Cancer Therapeutics Response Portal v2 / NCI CTD2, https://ctd2-data.nci.nih.gov/Public/Broad/CTRPv2.0_2015_ctd2_ExpandedDataset/CTRPv2.0_2015_ctd2_ExpandedDataset.zip; BHKLab-curated PSet on ORCESTRA, https://www.orcestra.ca/pset/10.5281/zenodo.3905470. See `system.file("extdata", "LICENSES.md", package = "AnnotationGx")` for bundled data provenance notes.

Examples

```
data(CTRP_sampleMetadata)
head(CTRP_sampleMetadata)
```

CTRP_treatmentMetadata *CTRP Treatment Metadata*

Description

A column-reduced example treatment metadata table derived from Cancer Therapeutics Response Portal v2 metadata.

Usage

```
data(CTRP_treatmentMetadata)
```

Format

A `data.table` with 545 rows and 2 columns:

CTRP.treatmentid CTRP compound name.

CTRP.broad_cpd_id Broad compound identifier.

Source

Cancer Therapeutics Response Portal v2 / NCI CTD2, https://ctd2-data.nci.nih.gov/Public/Broad/CTRPv2.0_2015_ctd2_ExpandedDataset/CTRPv2.0_2015_ctd2_ExpandedDataset.zip; BHKLab-curated PSet on ORCESTRA, https://www.orcestra.ca/pset/10.5281/zenodo.3905470. See `system.file("extdata", "LICENSES.md", package = "AnnotationGx")` for bundled data provenance notes.

Examples

```
data(CTRP_treatmentMetadata)
head(CTRP_treatmentMetadata)
```

DatasetInfo *BioMart Dataset Information Class*

Description

Class that stores information about a dataset within a BioMart database.

Details

This class encapsulates metadata about a specific dataset in a BioMart database, including its name, description, display name, and a reference to the parent mart.

Public fields

`name` The internal name of the dataset

`description` A textual description of the dataset

`displayName` The human-readable name of the dataset

`mart` The MartInfo object this dataset belongs to

Methods

Public methods:

- `DatasetInfo$new()`
- `DatasetInfo$print()`
- `DatasetInfo$clone()`

`DatasetInfo$new()`: Initialize a new DatasetInfo object

Usage:

```
DatasetInfo$new(name, description, displayName, mart)
```

Arguments:

`name` Character, the internal name of the dataset

`description` Character, the dataset description

`displayName` Character, the human-readable name

`mart` MartInfo object, the parent mart

Returns: A new DatasetInfo object

`DatasetInfo$print()`: Print method for DatasetInfo objects

Usage:

```
DatasetInfo$print(...)
```

Arguments:

`...` Additional arguments (not used)

Returns: Invisibly returns self

`DatasetInfo$clone()`: The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`FilterInfo` *BioMart Filter Information Class*

Description

Represents a filter from a BioMart dataset.

Details

This class encapsulates information about a filter available in a BioMart dataset. Filters are used to specify conditions for querying and subsetting data from BioMart.

Public fields

`name` The internal name of the filter used in BioMart queries

`displayName` The human-readable name of the filter shown in user interfaces

`description` A textual description explaining the filter's purpose

`type` The data type of the filter (e.g., "string", "boolean", "list")

`isHidden` Logical flag indicating if the filter should be hidden in user interfaces

`values` Possible values for the filter if it has a fixed/enumerated set of options

Methods

Public methods:

- FilterInfo$new()
- FilterInfo$print()
- FilterInfo$clone()

`FilterInfo$new()`: Initialize a new FilterInfo object

Usage:

```
FilterInfo$new(
  name,
  displayName = NULL,
  description = NULL,
  type = NULL,
  isHidden = NULL,
  values = NULL
)
```

Arguments:

`name` Character, the internal name of the filter used in API calls

`displayName` Character, the human-readable name shown to users

`description` Character, the filter description explaining its purpose

`type` Character, the data type of the filter (e.g., "string", "boolean")

`isHidden` Logical, whether the filter should be hidden in UIs

`values` List or vector, possible values for the filter if applicable

Returns: A new FilterInfo object

`FilterInfo$print()`: Print method for FilterInfo objects

Usage:

```
FilterInfo$print(...)
```

Arguments:

`...` Additional arguments passed to print methods (not used)

Returns: Invisibly returns self for method chaining

`FilterInfo$clone()`: The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

gCSI_sampleMetadata *gCSI Sample Metadata*

Description

A column-reduced example sample metadata table derived from Genentech Cell Line Screening Initiative (gCSI) metadata.

Usage

```
data(gCSI_sampleMetadata)
```

Format

A `data.table` with 1350 rows and 2 columns:

Characteristics.cell.line. gCSI cell line name.

Comment.ENA_SAMPLE. ENA sample accession.

Source

gCSI source metadata; BHKLab-curated PSet on ORCESTRA, https://www.orcestra.ca/pset/10.5281/zenodo.4737437. See `system.file("extdata", "LICENSES.md", package = "AnnotationGx")` for bundled data provenance notes.

Examples

```
data(gCSI_sampleMetadata)
head(gCSI_sampleMetadata)
```

gCSI_treatmentMetadata *gCSI Treatment Metadata*

Description

A column-reduced example treatment metadata table derived from gCSI response metadata.

Usage

```
data(gCSI_treatmentMetadata)
```

Format

A `data.table` with 44 rows and 2 columns:

gCSI.treatmentid gCSI treatment name.

gCSI.NormDrugName Normalized gCSI treatment name.

Source

gCSI source metadata; BHKLab-curated PSet on ORCESTRA, https://www.orcestra.ca/pset/10.5281/zenodo.4737437. See `system.file("extdata", "LICENSES.md", package = "AnnotationGx")` for bundled data provenance notes.

Examples

```
data(gCSI_treatmentMetadata)
head(gCSI_treatmentMetadata)
```

---

getChemblFilterTypes *Get the Chembl filter types*

Description

This function retrieves the Chembl filter types.

Usage

```
getChemblFilterTypes()
```

Value

A list of Chembl filter types.

Examples

```
getChemblFilterTypes()
```

---

getChemblMechanism *Get ChEMBL Mechanism*

Description

This function retrieves information about the mechanism of action for a given ChEMBL ID.

Usage

```
getChemblMechanism(
  chembl.ID,
  resources = "mechanism",
  field = "molecule_chembl_id",
  filter_type = "in",
  returnURL = FALSE,
  raw = FALSE
)
```

Arguments

chembl.ID	The ChEMBL ID of the molecule.
resources	The ChEMBL resource to query (default: "mechanism").
field	The field to filter on (default: "molecule_chembl_id").
filter_type	The filter type to use (default: "in").
returnURL	Logical indicating whether to return the constructed URL (default: FALSE).
raw	Logical indicating whether to return the raw response JSON (default: FALSE).

Value

A data.table containing the retrieved mechanism information.

Examples

```
# Requires the live ChEMBL service.
tryCatch({
  getChemblMechanism("CHEMBL1413")
  getChemblMechanism("CHEMBL1413",
    resources = "mechanism", field = "molecule_chembl_id",
    filter_type = "in", returnURL = FALSE, raw = FALSE
  )
},
httr2_failure = function(e) message("ChEMBL unavailable: ", conditionMessage(e)),
annotationgx_chembl_unavailable = function(e) {
  message("ChEMBL unavailable: ", conditionMessage(e))
})
```

---

getChemblResourceFields

Get the fields of a Chembl resource

Description

This function retrieves the fields of a Chembl resource.

Usage

```
getChemblResourceFields(resource)
```

Arguments

resource	The Chembl resource.

Value

A character vector containing the names of the fields.

Examples

```
# Requires the live ChEMBL service.
tryCatch({
  getChemblResourceFields("molecule")
},
httr2_failure = function(e) message("ChEMBL unavailable: ", conditionMessage(e)),
annotationgx_chembl_unavailable = function(e) {
  message("ChEMBL unavailable: ", conditionMessage(e))
})
```

---

getChemblResources *getChemblResources function*

Description

This function retrieves the Chembl resources.

Usage

```
getChemblResources()
```

Value

A list of Chembl resources.

Examples

```
getChemblResources()
```

---

getOncotreeMainTypes *Get the main types from the Oncotree database.*

Description

This function retrieves the main types from the Oncotree database.

Usage

```
getOncotreeMainTypes()
```

Value

A `data.table` containing the main types from the Oncotree database.

Examples

```
# Requires internet connection to Oncotree API
if (interactive()) {
  getOncotreeMainTypes()
}
```

getOncotreeTumorTypes *Get the tumor types from the Oncotree database.*

Description

This function retrieves the tumor types from the Oncotree database.

Usage

```
getOncotreeTumorTypes()
```

Value

A `data.table` containing the tumor types from the Oncotree database.

Examples

```
# Requires internet connection to Oncotree API
if (interactive()) {
  getOncotreeTumorTypes()
}
```

getOncotreeVersions *Get available Oncotree versions*

Description

This function retrieves the available versions of Oncotree.

Usage

```
getOncotreeVersions()
```

Value

A `data.table` containing available Oncotree versions.

Examples

```
# Requires internet connection to Oncotree API
if (interactive()) {
  getOncotreeVersions()
}
```

getPubchemAnnotationHeadings *Get annotation headings (name only) based on type and heading criteria.*

Description

Get annotation headings (name only) based on type and heading criteria.

Usage

```
getPubchemAnnotationHeadings(type = "all", heading = NULL)
```

Arguments

type	The type of annotation headings to retrieve. Options include "Compound", "Gene", "Taxonomy", "Element", "Assay", "Protein", "Cell", "Pathway", or "all" (default).
heading	The specific heading to filter the results by. Defaults to NULL, which retrieves all headings.

Value

A `data.table` containing the annotation headings and types.

Examples

```
getPubchemAnnotationHeadings()
getPubchemAnnotationHeadings(type = "Compound")
getPubchemAnnotationHeadings(heading = "ChEMBL*")
getPubchemAnnotationHeadings(type = "Compound", heading = "ChEMBL*")
```

getPubchemCompound *Retrieve PubChem compound information*

Description

This function retrieves compound information from PubChem using the PubChem REST API. Used by other functions to retrieve compound information.

Usage

```
getPubchemCompound(
  ids,
  from = "cid",
  to = "property",
  properties = c("Title", "InChIKey"),
  raw = FALSE,
  query_only = FALSE,
```

```
  output = "JSON",
  ...
)
```

Arguments

ids	A vector of compound identifiers.
from	The source namespace of the compound identifiers. Default is 'cid'.
to	The target namespace for the compound information. Default is 'property'.
properties	A character vector specifying the properties to retrieve.
raw	Logical indicating whether to return the raw query results. Default is FALSE.
query_only	Logical indicating whether to only perform the query without retrieving the results. Default is FALSE.
output	The format of the query results. Default is 'JSON'.
...	Additional arguments to be passed to the query_pubchem_rest function.

Value

A data.table containing the retrieved compound information.

Examples

```
properties <- c("Title", "MolecularFormula", "InChIKey", "CanonicalSMILES")
getPubchemCompound(c(3672, 176870), from = "cid", to = "property", properties = properties)
```

---

getPubchemProperties *Retrieves the PubChem XML schema and extracts property information.*

Description

This function retrieves the PubChem XML schema from the specified URL and extracts the property information from it. The property information includes the name and type of each property.

Usage

```
getPubchemProperties()
```

Value

A data table containing the extracted property information.

Examples

```
# Requires internet connection to PubChem
if (interactive()) {
  getPubchemProperties()
}
```

getPubchemStatus *Retrieves the status of a PubChem request*

Description

This function sends a request to PubChem to retrieve the status of a given URL. It returns the status code and, if specified, the parsed information from the response.

Usage

```
getPubchemStatus(
  returnMessage = FALSE,
  printMessage = TRUE,
 url = "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/Aspirin/cids/JSON"
)
```

Arguments

returnMessage	Logical indicating whether to return the parsed information from the response.
printMessage	Logical indicating whether to print the status message.
url	The URL to send the request to. Default is "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/nar

Value

The status code of the response. If returnMessage is TRUE, the parsed information from the response is also returned.

Examples

```
getPubchemStatus()
getPubchemStatus(returnMessage = TRUE)
getPubchemStatus(printMessage = FALSE)
```

getUnichemSources *Get the list of sources in UniChem.*

Description

Get the list of sources in UniChem.

Usage

```
getUnichemSources(all_columns = FALSE)
```

Arguments

all_columns	boolean Whether to return all columns. Defaults to FALSE. Returns a data.table with the following columns: • CompoundCount (integer): Total of compounds provided by that source • BaseURL (string): Source Base URL for compounds • Description (string): Source database description • LastUpdated (string): Date in which the source database was last updated • Name (string): Short name of the source database • NameLabel (string): Machine readable label name of the source database • NameLong (string): Full name of the source database • SourceID (integer): Unique ID for the source database • Details (string): Notes about the source • ReleaseDate (string): Date in which the source database was released • ReleaseNumber (integer): Release number of the source database data stored in UniChEM • URL (string): Main URL for the source • UpdateComments (string): Notes about the update process of that source to UniChEM

Value

A data.table with the list of sources in UniChem. Release metadata not provided by UniChem are returned as NA.

Examples

```
# Requires internet connection to UniChem
if (interactive()) {
  getUnichemSources()
}
```

---

mapCell2Accession *Map Cell Line IDs to Accession Numbers*

Description

This function maps cell line IDs to accession numbers using the Cellosaurus database.

Usage

```
mapCell2Accession(
  ids,
  numResults = 10000,
  from = "idsy",
  sort = "ac",
  keep_duplicates = FALSE,
  fuzzy = FALSE,
  query_only = FALSE,
```

```
  raw = FALSE,
  parsed = TRUE,
  include_query = TRUE,
  ...
)
```

Arguments

ids	A character vector of cell line IDs.
numResults	The maximum number of results to return for each query. Default is 1000.
from	The type of input IDs. Possible values are "idsy" (default), "ac", "id", "sy", and "misspelling".
sort	The sorting order of the results. Possible values are "ac" (default), "id", "sy", and "misspelling".
keep_duplicates	Logical indicating whether to keep duplicate results. Default is FALSE.
fuzzy	Logical indicating whether to perform a fuzzy search. Default is FALSE.
query_only	Logical indicating whether to return only the query URLs. Default is FALSE.
raw	Logical indicating whether to return the raw HTTP responses. Default is FALSE.
parsed	Logical indicating whether to parse the response text. Default is TRUE.
include_query	Logical indicating whether to include the query* columns (e.g. query, query:ac) in the returned result. Default is TRUE.
...	Currently unused. Reserved for future extensions.

Value

A data.table containing the mapped cell line IDs and accession numbers. When parsed = FALSE, the returned table also includes Cellosaurus metadata columns that have been renamed to user-friendly titles (for example, sy becomes synonyms).

Examples

```
mapCell2Accession(ids = c("A549", "MCF7"))
```

---

mapCID2Properties *Map PubChem Compound IDs to Properties*

Description

This function maps PubChem Compound IDs to specified properties using the PubChem REST API. See getPubchemProperties for a list of available properties.

Usage

```
mapCID2Properties(ids, properties, ...)
```

Arguments

ids	A vector of PubChem Compound IDs.
properties	A vector of property names to retrieve for each compound.
...	Additional arguments to be passed to the getPubchemCompound function.

Value

A data frame containing the mapped properties for each compound.

Examples

```
mapCID2Properties(ids = c(123, 456), properties = c("MolecularWeight", "CanonicalSMILES"))
```

---

mapCompound2CID *Map compound names to PubChem CIDs*

Description

This function maps compound names to PubChem CIDs using the PubChem REST API.

Usage

```
mapCompound2CID(names, first = FALSE, ...)
```

Arguments

names	A character vector of compound names.
first	Logical indicating whether to return only the first CID for each compound name (default is FALSE).
...	Additional arguments to be passed to the getPubchemCompound function.

Value

A character vector of PubChem CIDs.

Examples

```
mapCompound2CID(c("aspirin", "caffeine"))
```

`MartInfo` *BioMart Information Class*

Description

Class that stores information about a BioMart database.

Details

This class encapsulates metadata about a BioMart database, including its name, display name, description, configuration, and other properties.

Public fields

`name` The internal name of the mart

`displayName` The human-readable name of the mart

`description` A textual description of the mart

`config` Configuration identifier for the mart

`isHidden` Flag indicating if the mart is hidden

`operation` Operation mode of the mart

`meta` Additional metadata for the mart

`group` Group the mart belongs to

Methods

Public methods:

- `MartInfo$new()`
- `MartInfo$print()`
- `MartInfo$clone()`

`MartInfo$new()`**:** Initialize a new MartInfo object

Usage:

```
MartInfo$new(
  name,
  displayName,
  description,
  config,
  isHidden,
  operation,
  meta,
  group
)
```

Arguments:

`name` Character, the internal name of the mart

`displayName` Character, the human-readable name

`description` Character, the mart description

`config` Character, the configuration identifier

`isHidden` Logical, whether the mart is hidden

`operation` Character, the operation mode

`meta` List, additional metadata

`group` Character, the group the mart belongs to

Returns: A new MartInfo object

`MartInfo$print()`: Print method for MartInfo objects

Usage:

```
MartInfo$print(...)
```

Arguments:

`...` Additional arguments (not used)

Returns: Invisibly returns self

`MartInfo$clone()`: The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`matchNested` *Match inside nested elements*

Description

Match inside nested elements

Usage

```
matchNested(x, table, ..., keep_duplicates = FALSE)

## S4 method for signature 'character,list'
matchNested(x, table, keep_duplicates)

## S4 method for signature 'numeric,list'
matchNested(x, table, keep_duplicates)

## S4 method for signature 'character,data.table'
matchNested(x, table, keep_duplicates)

## S4 method for signature 'numeric,data.table'
matchNested(x, table, keep_duplicates)

## S4 method for signature 'character,data.frame'
matchNested(x, table, keep_duplicates)

## S4 method for signature 'character,data.frame'
matchNested(x, table, keep_duplicates)
```

Arguments

x	The values to be matched.
table	The values to be matched against. Applies across rows for DataFrame method.
...	Additional arguments to be passed to the method.
keep_duplicates	A logical value indicating whether to keep duplicates.

Details

Intentionally only performs exact matching. Refer to filterNested function for partial matching support with regular expressions.

Value

integer. A positional vector corresponding to values defined in table the same size as x.

Examples

```
showMethods("matchNested")
```

---

queryChemblAPI *Query the ChEMBL API*

Description

This function queries the ChEMBL API using the specified parameters and returns the response in JSON format.

Usage

```
queryChemblAPI(resource, field, filter_type, value, format = "json")
```

Arguments

resource	The resource to query in the ChEMBL API.
field	The field to filter on in the ChEMBL API.
filter_type	The type of filter to apply in the ChEMBL API.
value	The value to filter on in the ChEMBL API.
format	The format of the response (default is "json").

Value

The response from the ChEMBL API in JSON format.

Examples

```
# Requires the live ChEMBL service.
tryCatch({
  queryChemblAPI("mechanism", "molecule_chembl_id", "in", "CHEMBL1413")
},
httr2_failure = function(e) message("ChEMBL unavailable: ", conditionMessage(e)),
annotationgx_chembl_unavailable = function(e) {
  message("ChEMBL unavailable: ", conditionMessage(e))
})
```

---

queryUnichemCompound *Query UniChem for a compound.*

Description

This function queries the UniChem API for a compound based on the provided parameters.

Usage

```
queryUnichemCompound(
  compound,
  type,
  sourceID = NA_integer_,
  request_only = FALSE,
  raw = FALSE,
  progress = "Querying UniChem...",
  ...
)
```

Arguments

compound	character, integer, or a list of such values. When a vector or list is supplied, each element is queried and the results are returned as a named list.
type	character The type of compound identifier to search for. Valid types are "uci", "inchi", "inchikey", and "sourceID".
sourceID	integer The source ID to search for if the type is "sourceID". When querying multiple compounds, this can be a vector the same length as compound or a single value recycled to all queries. Defaults to NA.
request_only	boolean Whether to return the request only. Defaults to FALSE.
raw	boolean Whether to return the raw response. Defaults to FALSE.
progress	logical or character. Passed through to .perform_request_parallel() when multiple compounds are supplied. Use a character string to customise the progress label. Defaults to "Querying UniChem...".
...	Additional arguments.

Value

For a single query, a list with the external mappings and the UniChem mappings. For multiple queries, a named list of such results (one per compound). If raw = TRUE, raw responses are returned instead.

Examples

```
if (interactive()) {
  queryUnichemCompound(
    type = "sourceID",
    compound = "444795",
    sourceID = 22
  )
}
```

---

query_hgnc_by_genes *Query HGNC BioMart by Gene Symbols*

Description

Retrieves data from the HGNC BioMart for a list of gene symbols.

Usage

```
query_hgnc_by_genes(genes, attributes, mart_name = NULL, dataset_name = NULL)
```

Arguments

genes	Character vector of HGNC gene symbols.
attributes	Character vector of attribute display names to return.
mart_name	Character, optional name or display name of the mart to use (default: first available mart).
dataset_name	Character, optional name or display name of the dataset to use (default: first available dataset).

Details

This function connects to the HGNC BioMart service and retrieves specified attributes for a list of gene symbols. It automatically handles the selection of marts and datasets based on provided parameters.

Value

A data.table with results for matching gene symbols and attributes.

Examples

```
# Requires internet connection to HGNC BioMart
if (interactive()) {
  query_hgnc_by_genes(
    genes = c("TP53", "BRCA1"),
    attributes = c("Approved symbol", "Approved name")
  )
}
```

standardize_names *Standardize Names*

Description

This function takes a character vector and standardizes the names by converting them to lowercase, removing any trailing information after a comma, removing any information within square brackets or parentheses, removing any non-alphanumeric characters, replacing empty names with "invalid", and converting the names to uppercase.

Usage

```
standardize_names(object)
```

Arguments

object	A character vector containing the names to be standardized.

Value

A character vector with the standardized names.

Examples

```
standardize_names(c("John Doe", "Jane Smith (Manager)", "Alice, PhD"))
# Output: [1] "JOHNDOE" "JANESMITH" "ALICE"
```

strSplit *Split a character vector into a matrix based on a delimiter*

Description

This function splits a character vector into a matrix based on a specified delimiter. It can handle both finite and infinite splits.

Usage

```
strSplit(x, split, fixed = TRUE, n = Inf)
```

Arguments

x	A character vector to be split
split	A character string specifying the delimiter
fixed	A logical value indicating whether the delimiter should be treated as a fixed string
n	An integer specifying the maximum number of splits to be performed

Value

A matrix where each row represents a split element

Examples

```
strSplit("Hello,World", ",")
# Output:
#      [,1]    [,2]
# [1,] "Hello" "World"
```

`unlistNested` *Unlists a nested list and removes NA values and duplicates.*

Description

This function takes a nested list as input and unlists it recursively. It then removes any NA values and duplicates from the resulting vector.

Usage

```
unlistNested(element)
```

Arguments

`element`	The nested list to be unlisted.

Value

A vector with NA values and duplicates removed.

Examples

```
nested_list <- list(a = list(1, 2, NA), b = list(3, 4, 5))
unlistNested(nested_list)
# Output: [1] 1 2 3 4 5
```

Index

∗ **datasets**
  CCLE_sampleMetadata, 10
  CCLE_treatmentMetadata, 11
  CTRP_sampleMetadata, 13
  CTRP_treatmentMetadata, 13
  gCSI_sampleMetadata, 17
  gCSI_treatmentMetadata, 17
∗ **internal**
  AnnotationGx-package, 3
  AttributeInfo, 5
  AttributeSet, 6
  BioMartClient, 7
  bm_query_builder, 9
  DatasetInfo, 14
  FilterInfo, 15
  MartInfo, 28

annotateCellAccession, 3
annotatePubchemCompound, 4
AnnotationGx *(*AnnotationGx-package*)*, 3
AnnotationGx-package, 3
AttributeInfo, 5
AttributeSet, 6

BioMartClient, 7
bm_query_builder, 9

CCLE_sampleMetadata, 10
CCLE_treatmentMetadata, 11
cellosaurus_fields, 12
cellosaurusAPIVersion, 11
cleanCharacterStrings, 12
CTRP_sampleMetadata, 13
CTRP_treatmentMetadata, 13

DatasetInfo, 14

FilterInfo, 15

gCSI_sampleMetadata, 17
gCSI_treatmentMetadata, 17
getChemblFilterTypes, 18
getChemblMechanism, 18
getChemblResourceFields, 19
getChemblResources, 20
getOncotreeMainTypes, 20
getOncotreeTumorTypes, 21
getOncotreeVersions, 21
getPubchemAnnotationHeadings, 22
getPubchemCompound, 22
getPubchemProperties, 23
getPubchemStatus, 24
getUnichemSources, 24

mapCell2Accession, 25
mapCID2Properties, 26
mapCompound2CID, 27
MartInfo, 28
matchNested, 29
matchNested,character,data.frame-method
  *(*matchNested*)*, 29
matchNested,character,data.table-method
  *(*matchNested*)*, 29
matchNested,character,list-method
  *(*matchNested*)*, 29
matchNested,numeric,data.table-method
  *(*matchNested*)*, 29
matchNested,numeric,list-method
  *(*matchNested*)*, 29

query_hgnc_by_genes, 32
queryChemblAPI, 30
queryUnichemCompound, 31

standardize_names, 33
strSplit, 33

unlistNested, 34