Package 'Rpolyhedra'

January 20, 2025

Type Package

Title Polyhedra Database

Version 0.5.6

Language en-US

Maintainer Alejandro Baranek <abaranek@dc.uba.ar>

Description
A polyhedra database scraped from various sources as R6 objects and 'rgl' visualizing capabilities.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

Depends R (>= 3.5.0)

Imports R6, geometry, rgl, stringr, XML, digest, lgr, dplyr, jsonlite

Suggests testthat, knitr, pkgdown, rmarkdown, covr, codemetar

Collate 'Rpolyhedra-package.R' 'polyhedra-lib.R' 'ledger-lib.R'
'db-lib.R' 'env-lib.R' 'package-lib.R' 'serialization-lib.R'
'public-lib.R' 'test-lib.R' 'zzz.R'

BugReports https://github.com/ropensci/Rpolyhedra/issues

URL https://docs.ropensci.org/Rpolyhedra/,
https://github.com/ropensci/Rpolyhedra

StagedInstall TRUE

NeedsCompilation no

Author Alejandro Baranek [aut, com, cre, cph],
Leonardo Belen [aut, com, cph],
qbotics [cph],
Barret Schloerke [rev],
Lijia Yu [rev]

Repository CRAN

Date/Publication 2024-11-06 15:10:02 UTC

Contents

`Rpolyhedra-package` *Rpolyhedra: Polyhedra Database*

Description

A polyhedra database scraped from various sources as R6 objects and 'rgl' visualizing capabilities.

Details

A polyhedra database scraped from:

- http://paulbourke.net/dataformats/phd/: PHD files as R6 objects and 'rgl' visualizing capabilities. The PHD format was created to describe the geometric polyhedra definitions derived mathematically <https://netlib.org/polyhedra/> by Andrew Hume and by the Kaleido program of Zvi Har'El.
- http://dmccooey.com/Polyhedra/: Polyhedra text datafiles.

Author(s)

Maintainer: Alejandro Baranek <abaranek@dc.uba.ar> [compiler, copyright holder]

Authors:

- Leonardo Belen <leobelen@gmail.com> [compiler, copyright holder]

Other contributors:

- qbotics <qbotics6@gmail.com> [copyright holder]
- Barret Schloerke <schloerke@gmail.com> [reviewer]
- Lijia Yu <yu@lijiayu.net> [reviewer]

See Also

Useful links:

- https://docs.ropensci.org/Rpolyhedra/
- https://github.com/ropensci/Rpolyhedra
- Report bugs at https://github.com/ropensci/Rpolyhedra/issues

---

`genLogger` *genLogger*

Description

Returns a configured logger with threshold according r6 object. This function is usually called in class constructors

Usage

```
genLogger(r6.object)
```

Arguments

`r6.object`	an r6.object

Author(s)

ken4rab

getAvailablePolyhedra *Get available polyhedra*

Description

Gets the list of names of available polyhedra and its status in the polyhedra database, which can be later called with getPolyhedron

Usage

```
getAvailablePolyhedra(sources, search.string)
```

Arguments

sources	A string vector containing the source, which can be obtained from getAvailable-Sources().
search.string	A search string

Value

polyhedra names vector

See Also

getAvailableSources

Examples

```
# gets all polyhedra in the database
available.polyhedra <- getAvailablePolyhedra()

# returns all polyhedra from a given source, in this case, netlib
available.netlib.polyhedra <- getAvailablePolyhedra(sources = "netlib")

# search within the polyhedron names

cube <- getAvailablePolyhedra(sources = "netlib", search.string = "cube")
cube
```

getAvailableSources *Get available sources*

Description

Gets the list of names of available sources in database to be used later as references to the package.

Usage

```
getAvailableSources()
```

Value

sources string vector, which can be obtained from getAvailableSources()

See Also

getAvailablePolyhedra, getPolyhedron

Examples

```
# gets all sources in the database
available.sources <- getAvailableSources()

# returns all polyhedra from all sources
available.polyhedra <- getAvailablePolyhedra(sources = available.sources)

# search within the polyhedron names from all sources
cubes <- getAvailablePolyhedra(
  sources = available.sources,
  search.string = "cube"
)
cubes
```

getLogger *getLogger*

Description

Returns the configured lgr of an r6 object. If the object don't have a lgr or is not initialized returns an error

Usage

```
getLogger(r6.object)
```

Arguments

r6.object	an r6.object

Author(s)

ken4rab

---

getPolyhedraObject *Get a polyhedra object*

Description

Return the polyhedra database handler.

Usage

```
getPolyhedraObject()
```

Value

.polyhedra

See Also

PolyhedraDatabase

---

getPolyhedron *Get polyhedron*

Description

Gets a polyhedron from the database. It returns an R6 Class with all its characteristics and functions. The object returned, of type Polyhedron, allows to the user to get access to all the functionality provided.

Usage

```
getPolyhedron(source = "netlib", polyhedron.name)
```

Arguments

source	string vector, which can be obtained from getAvailableSources()
polyhedron.name	a valid name of a polyhedron in the database. Current names can be found with getAvailablePolyhedra()

Value

polyhedron R6 object

See Also

getAvailablePolyhedra, getAvailableSources

Examples

```
tetrahedron <- getPolyhedron(
  source = "netlib",
  polyhedron.name = "tetrahedron"
)

# returns name of polyhedra
tetrahedron$getName()

# polyhedron state
tetrahedron.state <- tetrahedron$getState()

# Johnson symbol and Schlafli symbol
tetrahedron.state$getSymbol()

# vertex data.frame
tetrahedron.state$getVertices()

# List of faces of solid representation (3D)
tetrahedron.state$getSolid()

# List of faces of net representation (2D)
tetrahedron.state$getNet()
```

---

loggerSetupFile *loggerSetupFile*

Description

loggerSetupFile

Usage

```
loggerSetupFile(log.file, default.threshold = "info", append = TRUE)
```

Arguments

log.file	log path for logging file
default.threshold	threshold for setting root. Default = "info"
append	if set to FALSE, cleanup all previous logs

Author(s)

kenarab

`mutate_cond` *mutate_cond*

Description

mutate_cond

Usage

```
mutate_cond(.data, condition, ..., envir = parent.frame())
```

Arguments

`.data`	data frame to apply the mutate
`condition`	condition to conditionally apply mutate
`...`	mutation function
`envir`	environment to apply condition

`PolyhedraDatabase` *Polyhedra database*

Description

Scrapes all polyhedra in data folder to save a representation which is accessible by the final users upon call to `getPolyhedron()`.

Public fields

`version` version of database file

`polyhedra.rds.file` path of rds database file

`sources.config` Sources configuration for scraping different sources

`ledger` rr ledger of scraping process

`logger` class logger

Methods

Public methods:

Method `new()`**:** Create a new PolyhedraDatabase object.

Usage:

```
PolyhedraDatabase$new()
```

Returns: A new 'PolyhedraDatabase' object.

Method `getVersion()`**:** get the version of the current object.

Usage:

```
PolyhedraDatabase$getVersion()
```

Returns: Database version

Method `configPolyhedraRDSPath()`**:** sets the path of the RDS object

Usage:

```
PolyhedraDatabase$configPolyhedraRDSPath()
```

Returns: Database version

Method `existsSource()`**:** Determines if the source exists on the database

Usage:

```
PolyhedraDatabase$existsSource(source)
```

Arguments:

`source` source description

Returns: boolean value

Method `addSourceConfig()`**:** add source.config to the database

Usage:

```
PolyhedraDatabase$addSourceConfig(source.config)
```

Arguments:

`source.config` SourceConfig object able to scrape source polyhedra definitions

Returns: PolyhedraDatabase object

Method `existsPolyhedron()`**:** Determines if the database includes a polyhedron which name matches the parameter value

Usage:

```
PolyhedraDatabase$existsPolyhedron(source = "netlib", polyhedron.name)
```

Arguments:

`source` source description

`polyhedron.name` polyhedron description

Returns: boolean value

Method `getPolyhedraSourceDir()`**:** gets polyhedra sources folder

Usage:

```
PolyhedraDatabase$getPolyhedraSourceDir(source, create.dir = TRUE)
```

Arguments:

`source` source description

`create.dir` if dir does not exists, create it

Returns: string with polyhedra sources path

Method `getPolyhedronFilename()`**:** gets the filename of the polyhedron matching parameter.

Usage:

```
PolyhedraDatabase$getPolyhedronFilename(source, polyhedron.name, extension)
```

Arguments:

`source` source description

`polyhedron.name` polyhedron description

`extension` extension of the polyhedron filename

Returns: string with polyhedron filename

Method `getPolyhedron()`**:** gets polyhedron object which name matches the parameter value

Usage:

```
PolyhedraDatabase$getPolyhedron(
  source = "netlib",
  polyhedron.name,
  strict = FALSE
)
```

Arguments:

`source` source description

`polyhedron.name` polyhedron description

`strict` halts execution if polyhedron not found

Returns: Polyhedron object

Method `addPolyhedron()`**:** add polyhedron object to the database

Usage:

```
PolyhedraDatabase$addPolyhedron(
  source = "netlib",
  source.filename,
  polyhedron,
  overwrite = FALSE,
  save.on.change = FALSE
)
```

Arguments:

`source` source description

`source.filename` filename of the polyhedron source definition

`polyhedron` polyhedron object

`overwrite` overwrite exiting definition

`save.on.change` saves Database state after operation

Returns: Polyhedron object

Method `configPolyhedraSource()`**:** Process parameter filenames using source.config parameter

Usage:

```
PolyhedraDatabase$configPolyhedraSource(
  source.config,
  source.filenames = NULL,
  max.quant = 0,
  save.on.change = FALSE
)
```

Arguments:

`source.config` source configuration for scraping files

`source.filenames` filenames of the polyhedron source definition

`max.quant` maximum filenames to process

`save.on.change` saves Database state after operation

Returns: Modified ‘PolyhedraDatabase‘ object.

Method `saveRDS()`**:** saveRDS

Usage:

```
PolyhedraDatabase$saveRDS(save.on.change = TRUE)
```

Arguments:

`save.on.change` saves Database state after operation

Returns: saveRDS return status

Method `cover()`**:** Cover objects and applies covering.code parameter

Usage:

```
PolyhedraDatabase$cover(
  mode,
  sources = names(self$sources.config),
  covering.code,
  polyhedra.names = NULL,
  max.quant = 0,
  save.on.change = FALSE,
  seed = NULL
)
```

Arguments:

`mode` covering mode. Available values are "scrape.queued", "scrape.retry","skipped", "test"

`sources` sources names

`covering.code` code for applying in covering

`polyhedra.names` polyhedra names to cover (optional)

`max.quant` maximum numbers of polyhedra to cover

`save.on.change` saves Database state after operation

`seed` seed for deterministic random generator

Returns: A list with resulting objects covered

Method `scrape()`**:** Scrape polyhedra queued sources

Usage:

```
PolyhedraDatabase$scrape(
  mode = "scrape.queued",
  sources = names(self$sources.config),
  max.quant = 0,
  time2scrape.source = 30,
  save.on.change = FALSE,
  skip.still.queued = FALSE
)
```

Arguments:

`mode` covering mode. Available values are "scrape.queued", "scrape.retry","skipped", "test"

`sources` sources names

`max.quant` maximum numbers of polyhedra to cover

`time2scrape.source` maximum time to spend scraping each source

`save.on.change` saves Database state after operation

`skip.still.queued` Flag unscraped files with status ‘skipped“

`covering.code` code for applying in covering

`polyhedra.names` polyhedra names to cover (optional)

Returns: A list with resulting objects covered

Method `testRR()`: testRR

Usage:

```
PolyhedraDatabase$testRR(sources = names(self$sources.config), max.quant = 0)
```

Arguments:

`sources` sources names

`max.quant` maximum numbers of polyhedra to cover

Returns: A list with resulting objects tested

Method `generateTestTasks()`: generate Test tasks for selected polyhedra

Usage:

```
PolyhedraDatabase$generateTestTasks(
  sources = names(self$sources.config),
  polyhedra.names = NULL,
  TestTaskClass,
  max.quant = 0
)
```

Arguments:

`sources` sources names

`polyhedra.names` polyhedra names to cover (optional)

`TestTaskClass` an R6 TestTaskClass class

`max.quant` maximum numbers of polyhedra to cover

Returns: A list with resulting TestTasks generated

Method `schedulePolyhedraSources()`: Schedules polyhedra sources for scraping

Usage:

```
PolyhedraDatabase$schedulePolyhedraSources(
  sources.config = getPackageEnvir(".available.sources"),
  source.filenames = NULL,
  max.quant = 0,
  save.on.change = FALSE
)
```

Arguments:

`sources.config` sources configurations for scraping files

`source.filenames` filenames of the polyhedron source definition

`max.quant` maximum filenames to process

`save.on.change` saves Database state after operation

Returns: Modified ‘PolyhedraDatabase‘ object.

Method `getAvailableSources()`: Returns available sources in current database

Usage:

```
PolyhedraDatabase$getAvailableSources()
```

Returns: A vector with names of available sources

Method `getAvailablePolyhedra()`**:** Retrieves all polyhedron within the source those names match with search.string

Usage:

```
PolyhedraDatabase$getAvailablePolyhedra(
  sources = self$getAvailableSources(),
  search.string = NULL,
  ignore.case = TRUE
)
```

Arguments:

`sources` sources names

`search.string` string for matching polyhedron names

`ignore.case` ignore case in search string

Returns: A list with resulting objects covered

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Polyhedron` *Polyhedron*

Description

Polyhedron container class, which is accessible by the final users upon call

Public fields

`file.id` Polyhedron file.id

`state` Polyhedron state

`logger` class logger

Methods

Public methods:

- `Polyhedron$new()`
- `Polyhedron$scrapeNetlib()`
- `Polyhedron$scrapeDmccooey()`
- `Polyhedron$deserialize()`
- `Polyhedron$getName()`
- `Polyhedron$getState()`
- `Polyhedron$getSolid()`

- Polyhedron$isChecked()
- Polyhedron$getRGLModel()
- Polyhedron$exportToXML()
- Polyhedron$getErrors()
- Polyhedron$checkProperties()
- Polyhedron$clone()

Method `new()`**:** Create a polyhedronState object

Usage:

```
Polyhedron$new(file.id, state = NULL)
```

Arguments:

`file.id` the file id

`state` polyhedron state object

Returns: A new Polyhedron object.

Method `scrapeNetlib()`**:** scrape Netlib polyhedron definition

Usage:

```
Polyhedron$scrapeNetlib(netlib.p3.lines)
```

Arguments:

`netlib.p3.lines` vector with netlib definition lines

Returns: A new PolyhedronStateDefined object.

Method `scrapeDmccooey()`**:** scrape Dmccooey polyhedron definition

Usage:

```
Polyhedron$scrapeDmccooey(polyhedra.dmccooey.lines)
```

Arguments:

`polyhedra.dmccooey.lines` vector with Dmccooey definition lines

Returns: A new PolyhedronStateDefined object.

Method `deserialize()`**:** deserialize a polyhedron state definition

Usage:

```
Polyhedron$deserialize(serialized.polyhedron)
```

Arguments:

`serialized.polyhedron` a serialized version of a polyhedron state

Returns: A new PolyhedronStateDefined object.

Method `getName()`**:** get Polyhedron name

Usage:

```
Polyhedron$getName()
```

Returns: string with polyhedron name

Method `getState()`**:** Gets polyhedron state

Usage:

```
Polyhedron$getState()
```

Returns: A new PolyhedronState object.

Method `getSolid()`**:** Gets a solid definition

Usage:

```
Polyhedron$getSolid()
```

Returns: A list of vertex vectors composing polyhedron faces.

Method `isChecked()`**:** checks Edges consistency

Usage:

```
Polyhedron$isChecked()
```

Returns: A boolean value

Method `getRGLModel()`**:** Return an 'rgl' model with an optional transformation described by transformation.matrix parameter

Usage:

```
Polyhedron$getRGLModel(transformation.matrix = NULL)
```

Arguments:

`transformation.matrix` transformation matrix parameter

Returns: An tmesh3d object

Method `exportToXML()`**:** exports an XML definition of current polyhedron

Usage:

```
Polyhedron$exportToXML()
```

Returns: A character object with the XML definition

Method `getErrors()`**:** returns the errors found when processing current polyhedron

Usage:

```
Polyhedron$getErrors()
```

Returns: a data.frame with polyhedron errors

Method `checkProperties()`**:** check properties of current polyhedron

Usage:

```
Polyhedron$checkProperties(expected.vertices, expected.faces)
```

Arguments:

`expected.vertices` expected vertices number

`expected.faces` expected faces number

Returns: Unmodified polyhedron object

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Author(s)

ken4rab

`PolyhedronState` *PolyhedronState*

Description

This abstract class provide the basis from which every polyhedron state class derivate.

Public fields

`source` polyhedron definition source

`file.id` polyhedron file id

`errors` Errors string

`logger` class logger

Methods

Public methods:

- `PolyhedronState$new()`
- `PolyhedronState$addError()`
- `PolyhedronState$scrape()`
- `PolyhedronState$getName()`
- `PolyhedronState$getSolid()`
- `PolyhedronState$checkEdgesConsistency()`
- `PolyhedronState$applyTransformationMatrix()`
- `PolyhedronState$buildRGL()`
- `PolyhedronState$exportToXML()`
- `PolyhedronState$clone()`

Method `new()`**:** Create a polyhedronState object

Usage:

```
PolyhedronState$new(source, file.id)
```

Arguments:

`source` the source file

`file.id` the file id

Returns: A new PolyhedronState object. '@description Adds an error to the error string and log it as info

Method `addError()`**:**

Usage:

```
PolyhedronState$addError(current.error)
```

Arguments:

`current.error` the error to add

Method `scrape()`**:** Scrapes the polyhedra folder files

Usage:

```
PolyhedronState$scrape()
```

Method `getName()`**:** Get Polyhedron name

Usage:

```
PolyhedronState$getName()
```

Returns: string with polyhedron name

Method `getSolid()`**:** Returns the object corresponding to the solid

Usage:

```
PolyhedronState$getSolid()
```

Method `checkEdgesConsistency()`**:** Checks edge consistency

Usage:

```
PolyhedronState$checkEdgesConsistency()
```

Method `applyTransformationMatrix()`**:** Apply transformation matrix to polyhedron

Usage:

```
PolyhedronState$applyTransformationMatrix(transformation.matrix)
```

Arguments:

`transformation.matrix` the transformation matrix to apply to the polyhedron

Method `buildRGL()`**:** Creates a 'rgl' representation of the object

Usage:

```
PolyhedronState$buildRGL(transformation.matrix)
```

Arguments:

`transformation.matrix` the transformation matrix to apply to the polyhedron

Method `exportToXML()`**:** Gets an XML representation out of the polyhedron object

Usage:

```
PolyhedronState$exportToXML()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Author(s)

ken4rab

PolyhedronStateDefined

PolyhedronStateDefined

Description

Polyhedron State scraped and defined

Super class

Rpolyhedra::PolyhedronState -> PolyhedronStateDefined

Public fields

`file.id` polyhedron filename in original

`source` polyhedron definition source (netlib|dmccooey)

`name` polyhedron name (netlib|dmccooey)

`symbol` the eqn(1) input for two symbols separated by a tab; the Johnson symbol, and the Schlafli symbol (netlib)

`dual` the name of the dual polyhedron optionally followed by a horizontal tab and the number of the dual (netlib)

`sfaces` polyhedron solid face list (netlib)

`svertices` polyhedron solid vertices list (netlib)

`vertices` Polyhedron vertices list (netlib|dmccooey)

`vertices.centered` centered vertices for applying transformation matrices

`net` polyhedron 2D net model with vertices defined for a planar representation (netlib)

`solid` polyhedron list of edges which generate a solid (netlib|dmccooey)

`hinges` Polyhedron hinge list (netlib)

`dih` Dih attribute (netlib)

`edges` polyhedron edges (netlib|dmccooey)

`transformation.matrix` transformation matrix for calculations and visualizing polyhedron

Methods

Public methods:

- `PolyhedronStateDefined$new()`
- `PolyhedronStateDefined$scrape()`
- `PolyhedronStateDefined$saveToJson()`
- `PolyhedronStateDefined$getName()`
- `PolyhedronStateDefined$getSymbol()`
- `PolyhedronStateDefined$adjustVertices()`

- PolyhedronStateDefined$getVertices()
- PolyhedronStateDefined$getNet()
- PolyhedronStateDefined$getSolid()
- PolyhedronStateDefined$inferEdges()
- PolyhedronStateDefined$checkEdgesConsistency()
- PolyhedronStateDefined$triangulate()
- PolyhedronStateDefined$getConvHull()
- PolyhedronStateDefined$calculateMassCenter()
- PolyhedronStateDefined$getNormalizedSize()
- PolyhedronStateDefined$getTransformedVertices()
- PolyhedronStateDefined$resetTransformationMatrix()
- PolyhedronStateDefined$applyTransformationMatrix()
- PolyhedronStateDefined$buildRGL()
- PolyhedronStateDefined$exportToXML()
- PolyhedronStateDefined$expectEqual()
- PolyhedronStateDefined$serialize()
- PolyhedronStateDefined$clone()

Method `new()`**:** object initialization routine

Usage:

```
PolyhedronStateDefined$new(
  source,
  file.id,
  name,
  vertices,
  solid,
  net = NULL,
  symbol = "",
  dual = NULL,
  sfaces = NULL,
  svertices = NULL,
  hinges = NULL,
  dih = NULL,
  normalize.size = TRUE
)
```

Arguments:

`source` the library to use

`file.id` identifier of the definition file.

`name` the polyhedron name

`vertices` the vertices

`solid` the solid object

`net` the net

`symbol` the symbol

`dual` whether it is dual or not

`sfaces` the solid faces
`svertices` the solid vertices
`hinges` the hinges
`dih` the dih
`normalize.size` whether it has to normalize the size or not

Returns: A new PolyhedronStateDefined object.

Method `scrape()`**:** scrape polyhedron. As the state is defined this functions do nothing

Usage:

```
PolyhedronStateDefined$scrape()
```

Returns: current object

Method `saveToJson()`**:** saves the state to a JSON file

Usage:

```
PolyhedronStateDefined$saveToJson(pretty = FALSE)
```

Arguments:

`pretty` whether json output is pretty or not

Returns: a json object

Method `getName()`**:** get Polyhedron name

Usage:

```
PolyhedronStateDefined$getName()
```

Returns: string with polyhedron name

Method `getSymbol()`**:** get Polyhedron symbol

Usage:

```
PolyhedronStateDefined$getSymbol()
```

Returns: string with polyhedron symbol

Method `adjustVertices()`**:** adjust polyhedron Vertices

Usage:

```
PolyhedronStateDefined$adjustVertices(normalize.size = TRUE)
```

Arguments:

`normalize.size` whether it has to normalize the size or not

Returns: modified PolyhedronStateDefined object.

Method `getVertices()`**:** Get the polyhedron state

Usage:

```
PolyhedronStateDefined$getVertices(solid = FALSE)
```

Arguments:

`solid` toggles the production of solid vertices.

Method `getNet()`: Gets the net property

Usage:

```
PolyhedronStateDefined$getNet()
```

Method `getSolid()`: Gets the solid property

Usage:

```
PolyhedronStateDefined$getSolid()
```

Method `inferEdges()`: Infer edges

Usage:

```
PolyhedronStateDefined$inferEdges(force.recalculation = FALSE)
```

Arguments:

`force.recalculation` forces the recalculation of the edges

Method `checkEdgesConsistency()`: Checks edges consistency

Usage:

```
PolyhedronStateDefined$checkEdgesConsistency()
```

Method `triangulate()`: Triangulates the polyhedron

Usage:

```
PolyhedronStateDefined$triangulate(force = FALSE)
```

Arguments:

`force` forces the triangulation.

Method `getConvHull()`: Gets the convex hull

Usage:

```
PolyhedronStateDefined$getConvHull(
  transformation.matrix = self$transformation.matrix,
  vertices.id.3d = private$vertices.id.3d
)
```

Arguments:

`transformation.matrix` the transformation matrix

`vertices.id.3d` the vertices ids

Returns: the convex hull

Method `calculateMassCenter()`: Calculates the center of mass.

Usage:

```
PolyhedronStateDefined$calculateMassCenter(
  vertices.id.3d = private$vertices.id.3d,
  applyTransformation = TRUE
)
```

Arguments:

`vertices.id.3d` the vertices ids

`applyTransformation` does it need to apply transformations?

Method `getNormalizedSize()`**:** Gets the normalized size

Usage:

```
PolyhedronStateDefined$getNormalizedSize(size)
```

Arguments:

`size` the object's size

Method `getTransformedVertices()`**:** Gets the transformed vertices

Usage:

```
PolyhedronStateDefined$getTransformedVertices(
  vertices = self$vertices.centered,
  transformation.matrix = self$transformation.matrix
)
```

Arguments:

`vertices` input vertices

`transformation.matrix` the transformation matrix

Method `resetTransformationMatrix()`**:** Resets the transformation matrix

Usage:

```
PolyhedronStateDefined$resetTransformationMatrix()
```

Method `applyTransformationMatrix()`**:** Apply transformation matrix to polyhedron

Usage:

```
PolyhedronStateDefined$applyTransformationMatrix(transformation.matrix)
```

Arguments:

`transformation.matrix` the transformation matrix to apply to the polyhedron

Returns: an applied transformation.matrix

Method `buildRGL()`**:** Build 'rgl'

Usage:

```
PolyhedronStateDefined$buildRGL(transformation.matrix = NULL)
```

Arguments:

`transformation.matrix` the transformation matrix

Method `exportToXML()`**:** Exports the object to XML format

Usage:

```
PolyhedronStateDefined$exportToXML()
```

Method `expectEqual()`**:** Determines if a polyhedron is equal to this one.

Usage:

```
PolyhedronStateDefined$expectEqual(polyhedron)
```

Arguments:

`polyhedron` the polyhedron to compare to.

Method `serialize()`**:** Serialize the object.

Usage:

```
PolyhedronStateDefined$serialize()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Author(s)

ken4rab

`PolyhedronStateDeserializer` *PolyhedronStateDeserializer*

Description

Polyhedron state for deserialize from database

Super class

`Rpolyhedra::PolyhedronState` -> `PolyhedronStateDeserializer`

Public fields

`serialized.polyhedron` polyhedron definition serialized

Methods

Public methods:

- `PolyhedronStateDeserializer$new()`
- `PolyhedronStateDeserializer$scrape()`
- `PolyhedronStateDeserializer$clone()`

Method `new()`**:** Initialize PolyhedronStateDeserializer object

Usage:

```
PolyhedronStateDeserializer$new(serialized.polyhedron)
```

Arguments:

`serialized.polyhedron` a serialized polyhedron

Returns: A new PolyhedronStateDeserializer object.

Method `scrape()`**:** Generates a PolyhedronStateDefined from a serialized polyhedron

Usage:

```
PolyhedronStateDeserializer$scrape()
```

Returns: A new PolyhedronStateDefined object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Author(s)

ken4rab

`PolyhedronStateDmccooeyScraper`

PolyhedronStateDmccooeyScraper

Description

Scrapes polyhedra from a dmccooey file format

Super class

`Rpolyhedra::PolyhedronState` -> `PolyhedronStateDmccooeyScraper`

Public fields

`regexp.values.names` regexp for scraping values names

`regexp.rn` regexp for scraping real numbers

`regexp.values` regexp for scraping values

`regexp.vertex` regexp for scraping vertices

`regexp.faces` regexp for scraping faces

`polyhedra.dmccooey.lines` dmccooey polyhedra definition lines

`labels.map` labels map where values are

`values` labels map where values are

`vertices` specification

`vertices.replaced` 3D values

`faces` definition

Methods

Public methods:

- PolyhedronStateDmccooeyScraper$new()
- PolyhedronStateDmccooeyScraper$setupRegexp()
- PolyhedronStateDmccooeyScraper$scrapeValues()
- PolyhedronStateDmccooeyScraper$scrapeVertices()
- PolyhedronStateDmccooeyScraper$scrapeFaces()
- PolyhedronStateDmccooeyScraper$scrape()
- PolyhedronStateDmccooeyScraper$getName()
- PolyhedronStateDmccooeyScraper$applyTransformationMatrix()
- PolyhedronStateDmccooeyScraper$buildRGL()
- PolyhedronStateDmccooeyScraper$exportToXML()
- PolyhedronStateDmccooeyScraper$clone()

Method `new()`**:** Initialize Dmccooey scraper

Usage:

```
PolyhedronStateDmccooeyScraper$new(file.id, polyhedra.dmccooey.lines)
```

Arguments:

`file.id` identifier of the definition file.

`polyhedra.dmccooey.lines` raw Dmccooey definition file lines

Returns: A new PolyhedronStateDmccooeyScraper object.

Method `setupRegexp()`**:** setupRegexp for Dmccooey definition

Usage:

```
PolyhedronStateDmccooeyScraper$setupRegexp()
```

Returns: This PolyhedronStateDmccooeyScraper object with regexp defined.

Method `scrapeValues()`**:** scrape values from Dmccooey definition

Usage:

```
PolyhedronStateDmccooeyScraper$scrapeValues(values.lines)
```

Arguments:

`values.lines` values definitions in Dmccooey source

Returns: This PolyhedronStateDmccooeyScraper object with values defined.

Method `scrapeVertices()`**:** scrape polyhedron vertices from definition

Usage:

```
PolyhedronStateDmccooeyScraper$scrapeVertices(vertices.lines)
```

Arguments:

`vertices.lines` vertices definitions in Dmccooey source

Returns: This PolyhedronStateDmccooeyScraper object with faces defined.

Method `scrapeFaces()`**:** scrape polyhedron faces from definition

Usage:

```
PolyhedronStateDmccooeyScraper$scrapeFaces(faces.lines)
```

Arguments:

`faces.lines` face

Returns: This PolyhedronStateDmccooeyScraper object with faces defined.

Method `scrape()`**:** scrape Dmccooey polyhedron definition

Usage:

```
PolyhedronStateDmccooeyScraper$scrape()
```

Returns: A new PolyhedronStateDefined object.

Method `getName()`**:** get Polyhedron name

Usage:

```
PolyhedronStateDmccooeyScraper$getName()
```

Returns: string with polyhedron name

Method `applyTransformationMatrix()`**:** Apply transformation matrix to polyhedron

Usage:

```
PolyhedronStateDmccooeyScraper$applyTransformationMatrix(transformation.matrix)
```

Arguments:

`transformation.matrix` the transformation matrix to apply to the polyhedron

Method `buildRGL()`**:** Creates a 'rgl' representation of the object

Usage:

```
PolyhedronStateDmccooeyScraper$buildRGL(transformation.matrix)
```

Arguments:

`transformation.matrix` the transformation matrix to apply to the polyhedron

Method `exportToXML()`**:** serializes object in XML

Usage:

```
PolyhedronStateDmccooeyScraper$exportToXML()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Author(s)

ken4rab

PolyhedronStateNetlibScraper
PolyhedronStateNetlibScraper

Description

Scrapes polyhedra from a PHD file format.

Super class

`Rpolyhedra::PolyhedronState` -> `PolyhedronStateNetlibScraper`

Public fields

`netlib.p3.lines` The path to the PHD files

`labels.rows` Labels - row of appearance

`labels.map` Labels - Map of content

`errors` the errors found

Methods

Public methods:

- `PolyhedronStateNetlibScraper$new()`
- `PolyhedronStateNetlibScraper$extractRowsFromLabel()`
- `PolyhedronStateNetlibScraper$getLabels()`
- `PolyhedronStateNetlibScraper$scrapeNet()`
- `PolyhedronStateNetlibScraper$extractCFOutBrackets()`
- `PolyhedronStateNetlibScraper$scrapeVertices()`
- `PolyhedronStateNetlibScraper$setupLabelsOrder()`
- `PolyhedronStateNetlibScraper$getDataFromLabel()`
- `PolyhedronStateNetlibScraper$getName()`
- `PolyhedronStateNetlibScraper$scrape()`
- `PolyhedronStateNetlibScraper$applyTransformationMatrix()`
- `PolyhedronStateNetlibScraper$buildRGL()`
- `PolyhedronStateNetlibScraper$exportToXML()`
- `PolyhedronStateNetlibScraper$clone()`

Method `new()`**:** Initializes the object, taking the file.id and PDH file as parameters

Usage:

```
PolyhedronStateNetlibScraper$new(file.id, netlib.p3.lines)
```

Arguments:

`file.id` the file id

`netlib.p3.lines` the lines to add

Returns: A new PolyhedronStateNetlibScraper object.

Method `extractRowsFromLabel()`**:** Extracts data from the label, taking the label number and the expected label as parameters

Usage:

```
PolyhedronStateNetlibScraper$extractRowsFromLabel(label.number, expected.label)
```

Arguments:

`label.number` the label number

`expected.label` the expected label

Method `getLabels()`**:** get Labels from current netlib file description

Usage:

```
PolyhedronStateNetlibScraper$getLabels()
```

Returns: a list containing labels from netlib file description

Method `scrapeNet()`**:** scrape Net Model from netlib format

Usage:

```
PolyhedronStateNetlibScraper$scrapeNet(net.txt, offset = 0)
```

Arguments:

`net.txt` a vector containing net model in netlib format

`offset` in numbering vertices

Returns: a list containing a net model

Method `extractCFOutBrackets()`**:** Remove brackets for current field content

Usage:

```
PolyhedronStateNetlibScraper$extractCFOutBrackets(x)
```

Arguments:

`x` a string containing brackets

Returns: value

Method `scrapeVertices()`**:** scrape vertices described in netlib format

Usage:

```
PolyhedronStateNetlibScraper$scrapeVertices(vertices.txt)
```

Arguments:

`vertices.txt` vector containing netlib format vertices

Returns: data.frame containing netlib vertices

Method `setupLabelsOrder()`**:** setupLabelsOrder

Usage:

```
PolyhedronStateNetlibScraper$setupLabelsOrder()
```

Arguments:

`vertices.txt` vector containing netlib format vertices

Returns: data.frame containing netlib vertices

Method `getDataFromLabel()`**:** Get data from label specified as parameter

Usage:

```
PolyhedronStateNetlibScraper$getDataFromLabel(label)
```

Arguments:

`label` the label to get data from

Returns: value

Method `getName()`**:** get Polyhedron name

Usage:

```
PolyhedronStateNetlibScraper$getName()
```

Returns: string with polyhedron name

Method `scrape()`**:** scrape Netlib polyhedron definition

Usage:

```
PolyhedronStateNetlibScraper$scrape()
```

Returns: A new PolyhedronStateDefined object.

Method `applyTransformationMatrix()`**:** Apply transformation matrix to polyhedron

Usage:

```
PolyhedronStateNetlibScraper$applyTransformationMatrix(transformation.matrix)
```

Arguments:

`transformation.matrix` the transformation matrix to apply to the polyhedron

Method `buildRGL()`**:** Creates a 'rgl' representation of the object

Usage:

```
PolyhedronStateNetlibScraper$buildRGL(transformation.matrix)
```

Arguments:

`transformation.matrix` the transformation matrix to apply to the polyhedron

Method `exportToXML()`**:** serializes object in XML

Usage:

```
PolyhedronStateNetlibScraper$exportToXML()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Author(s)

ken4rab

polyhedronToXML *Polyhedron to XML*

Description

Gets an XML representation out of the polyhedron object

Usage

```
polyhedronToXML(polyhedron.state.defined, is.transformed.vertices = TRUE)
```

Arguments

polyhedron.state.defined	the polyhedron to get a representation from
is.transformed.vertices	flag which states if vertices are in original position or transformationMatrix applied

Value

an XML document, ready to be converted to String with XML::saveXML()

Examples

```
# get the representation of a cube (netlib library)
XML::saveXML(polyhedronToXML(getPolyhedron("netlib", "cube")$state))
```

scrapePolyhedra *Scrape polyhedra objects*

Description

Gets polyhedra objects from text files of different sources, scheduling and scraping using predefined configurations.

Usage

```
scrapePolyhedra(
  scrape.config,
  source.filenames = NULL,
  sources.config = getUserEnvir(".available.sources"),
  logger = lgr
)
```

Arguments

scrape.config	predefined configuration for scraping
source.filenames	if not null specify which source filenames to scrape
sources.config	the sources that will be used by the function
logger	logger for inheriting threshold from calling class/function

Value

polyhedra db object

---

scrapePolyhedraSources *Scrape polyhedra sources*

Description

Scrapes polyhedra objects from text files of different sources, in order to make them available to the package.

Usage

```
scrapePolyhedraSources(sources.config =
         getUserEnvir(".available.sources"),
    max.quant.config.schedule = 0,
    max.quant.scrape = 0, time2scrape.source = 30,
    source.filenames = NULL, retry.scrape = FALSE,
    logger = lgr)
```

Arguments

sources.config	the sources that will be used by the function
max.quant.config.schedule	number of files to schedule
max.quant.scrape	number of files scrape
time2scrape.source	time applied to scrape source
source.filenames	if not null specify which source filenames to scrape
retry.scrape	should it retry scrape?
logger	logger for inheriting threshold from calling class/function

Value

polyhedra db object

`switchToFullDatabase` *Switch to full database*

Description

Prompts user for changing database to fulldb in user filespace. Also, allows the user to switch back to the package database, which is a minimal one for testing purposes.

Usage

```
switchToFullDatabase(env = NA, logger = lgr)
```

Arguments

`env`	The environment to run on, can be PACKAGE,
`logger`	logger for inheriting threshold from calling class/function HOME or NA. If NA, it asks the user for a an Environment.

Value

.data.env

Index