Package 'rBiopaxParser'

November 12, 2026

Type Package

Title Parses BioPax files and represents them in R

Version 2.52.0

Date 2020-07-14

Author Frank Kramer

Maintainer Frank Kramer <frank.kramer@informatik.uni-augsburg.de>

Description Parses BioPAX files and represents them in R, at the moment BioPAX level 2 and level 3 are supported.

License GPL (>= 2)

Depends R (>= 4.0), data.table

Imports XML

Suggests Rgraphviz, RCurl, graph, RUnit, BiocGenerics, RBGL, igraph

URL https://github.com/frankkramer-lab/rBiopaxParser

biocViews DataRepresentation

RoxygenNote 7.1.1

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

git_branch RELEASE_3_23

git_last_commit fca0061

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

rBiopaxParser-package 3
addBiochemicalReaction 4
addBiopaxInstance 5
addBiopaxInstances 6
addControl 6

addhash . . . 8
addns . . . 8
addPathway . . . 9
addPathwayComponents . . . 10
addPhysicalEntity . . . 11
addPhysicalEntityParticipant . . . 12
addPropertiesToBiopaxInstance . . . 13
biopax . . . 13
calcGraphOverlap . . . 14
checkValidity . . . 15
CLASS_INHERITANCE_BP2 . . . 15
CLASS_INHERITANCE_BP3 . . . 16
CLASS_PROPERTIES_BP2 . . . 16
CLASS_PROPERTIES_BP3 . . . 17
colorGraphNodes . . . 18
combineNodes . . . 19
createBiopax . . . 19
DATABASE_BIOPAX . . . 20
diffGraphs . . . 21
downloadBiopaxData . . . 22
generateNewUniqueID . . . 23
getClassProperties . . . 23
getInstanceClass . . . 24
getInstanceProperty . . . 25
getNeighborhood . . . 26
getParticipants . . . 26
getReferencedIDs . . . 27
getReferencingIDs . . . 28
getSubClasses . . . 29
getSuperClasses . . . 29
getXrefAnnotations . . . 30
hasProperty . . . 31
internal_checkArguments . . . 32
internal_generateXMLfromBiopax . . . 33
internal_getBiopaxModelAsDataFrame . . . 33
internal_NrOfXMLNodes . . . 34
internal_propertyListToDF . . . 34
internal_resolvePhysicalEntityParticipant . . . 35
internal_XMLInstance2DF . . . 36
intersectGraphs . . . 37
isOfClass . . . 37
isOfNamespace . . . 38
isURL . . . 39
layoutRegulatoryGraph . . . 39
listComplexComponents . . . 40
listInstances . . . 41
listInteractionComponents . . . 42
listPathwayComponents . . . 43

listPathways 44
mergePathways 45
pathway2AdjacancyMatrix 46
pathway2Geneset 47
pathway2Graph 48
pathway2RegulatoryGraph 50
plotRegulatoryGraph 51
print.biopax 52
readBiopax 52
removeDisconnectedParts 53
removeInstance 54
removeNodes 55
removeProperties 56
selectInstances 56
splitComplex 58
striphash 59
stripns 59
transitiveClosure 60
transitiveReduction 61
unfactorize 61
uniteGraphs 62
writeBiopax 63

Index **65**

`rBiopaxParser-package` *Parses BioPax level files and represents them in R*

Description

Parses BioPax files and represents them in R

Details

rBiopaxParser is a...

Package:	rBiopaxParser
Type:	Package
Version:	0.15
Date:	2012-08-22
License:	GPL (>= 2)

Author(s)

Frank Kramer <dev@frankkramer.de>

Examples

```
## Not run: biopax = readBiopax(file="biopaxmodel.owl")
```

---

addBiochemicalReaction *This function adds a new biochemical reaction to the biopax model.*

Description

This function adds a new biochemical reaction of class biochemicalReaction to the biopax model. This is a convenience function, internally the function addBiopaxInstance is called with properties LEFT and RIGHT set.

Usage

```
addBiochemicalReaction(biopax, LEFT = c(), RIGHT = c(), id = NULL)
```

Arguments

biopax	A biopax model
LEFT	vector of strings. IDs of the physicalEntityParticipant instances that are on the left side of this reaction.
RIGHT	vector of strings. IDs of the physicalEntityParticipant instances that are on the right side of this reaction.
id	string. ID for the control. If NULL a new ID is generated with prefix "biochemicalReaction".

Value

Returns the biopax model with the added pathway.

Author(s)

fkramer

Examples

```
biopax = createBiopax(level=2)
biopax = addPhysicalEntity(biopax, class="protein", id="p_id1", NAME="protein1")
biopax = addPhysicalEntityParticipant(biopax, "p_id1", id="PEP_p_id1")
biopax = addPhysicalEntity(biopax, class="protein", id="p_id2", NAME="protein2")
biopax = addPhysicalEntityParticipant(biopax, "p_id2", id="PEP_p_id2")
biopax = addBiochemicalReaction(biopax, LEFT=c("PEP_p_id1"), RIGHT=c("PEP_p_id2"), id="biochem_id_1")
biopax$dt
```

addBiopaxInstance *This function adds a new instance to an existing biopax model.*

Description

This function adds a new instance to an existing biopax model. "properties" is a named list of vectors, with the vector name as the name of the property and every entry of the vector a property value. Please note: case sensitivity! In Biopax Level 2 all properties are written in all capital letters. This will change in Biopax Level 3.

Usage

```
addBiopaxInstance(
  biopax,
  class,
  id,
  properties = list(NAME = c()),
  verbose = TRUE
)
```

Arguments

biopax	A biopax model
class	string. Class name
id	string. ID of the instance
properties	named list of properties.
verbose	logical. Be verbose about what was added.

Value

Returns the supplied biopax model with the new instance added.

Author(s)

Frank Kramer

Examples

```
biopax = createBiopax(level=2)
biopax = addBiopaxInstance(biopax, class="protein", id="id1", properties=list(NAME="protein1",SYNONYMS="p1"))
biopax$dt
```

addBiopaxInstances *This function adds new instances to an existing biopax model.*

Description

This function adds new instances (supplied as a compatible data.table) to an existing biopax model via rbind. Usually you want to start out at createBiopax and addPhysicalEntity and work your way up the ontology ladder.

Usage

```
addBiopaxInstances(biopax, newInstancesDF)
```

Arguments

biopax	A biopax model
newInstancesDF	data.table or data.frame. Must be compatible with internal biopax implementation.

Value

Returns the supplied biopax model with the new instances added.

Author(s)

Frank Kramer

Examples

```
# load data
 data(biopaxexample)
 biopax_temp = createBiopax(level=2)
biopax_temp = addBiopaxInstance(biopax_temp, class="protein", id="id1", properties=list(NAME="protein1",SYNONYM
 selectInstances(biopax_temp)
 biopax = addBiopaxInstances(biopax, selectInstances(biopax_temp))
```

addControl *This function adds a new control to the biopax model.*

Description

This function adds a new interaction of class control to the biopax model. This is a convenience function to add controls, internally the function addBiopaxInstance is called with properties CONTROL-TYPE, CONTROLLER and CONTROLLED set.

Usage

```
addControl(
  biopax,
  CONTROL_TYPE = c("ACTIVATION", "INHIBITION"),
  CONTROLLER = "",
  CONTROLLED = c(),
  id = NULL
)
```

Arguments

biopax	A biopax model
CONTROL_TYPE	string. Specifies wether this is an activating or inhibiting control.
CONTROLLER	string. ID of the physicalEntityParticipant instance that is the controller of this interaction.
CONTROLLED	vector of strings. IDs of the interaction and/or pathway instances that are being controlled.
id	string. ID for the control. If NULL a new ID is generated with prefix "control".

Value

Returns the biopax model with the added pathway.

Author(s)

fkramer

Examples

```
biopax = createBiopax(level=2)
biopax = addPhysicalEntity(biopax, class="protein", id="p_id1", NAME="protein1")
biopax = addPhysicalEntityParticipant(biopax, "p_id1", id="PEP_p_id1")
biopax = addPhysicalEntity(biopax, class="protein", id="p_id2", NAME="protein2")
biopax = addPhysicalEntityParticipant(biopax, "p_id2", id="PEP_p_id2")
biopax = addBiochemicalReaction(biopax, LEFT=c("PEP_p_id1"), RIGHT=c("PEP_p_id2"), id="biochem_id_1")
biopax = addPhysicalEntity(biopax, class="protein", id="p_id3", NAME="controllerProtein1")
biopax = addPhysicalEntityParticipant(biopax, "p_id3", id="PEP_p_id3")
biopax = addControl(biopax, CONTROL_TYPE="ACTIVATION", CONTROLLER="PEP_p_id3", CONTROLLED="biochem_id_1", id="c_
biopax$dt
```

addhash *Adds a hash in front of a string*

Description

Adds a hash in front of a string

Usage

```
addhash(x)
```

Arguments

x	A string to be preceeded by a hash

Value

The supplied string with a hash "#" pasted in front of it.

Author(s)

Frank Kramer

addns *Add a namespace tag to the supplied classname string*

Description

This function takes the input classname, checks if it already has a namespace, and if not pastes the namespace tag with a dividing ":" in front of it.

Usage

```
addns(classname, namespace = "bp")
```

Arguments

classname	A string containing a classname
namespace	A string containing a namespace

Value

If the classname is not preceeded by a namespace yet, the supplied namespace is pasted in front of it and returned.

Author(s)

Frank Kramer

addPathway *This function adds a new pathway to the biopax model.*

Description

This function adds a new pathway + its PATHWAY-COMPONENTS (references to interaction/pathways/pathwaySteps)

Usage

```
addPathway(
  biopax,
  NAME,
  PATHWAY_COMPONENTS = c(),
  id = NULL,
  ORGANISM = NULL,
  COMMENT = NULL
)
```

Arguments

biopax	A biopax model
NAME	string. Name of the pathway
PATHWAY_COMPONENTS	character vector. IDs of the pathway components. This must be IDs of instances of type interaction/pathway/pathwayStep (or their subclasses).
id	string. ID for the pathway. If NULL a new ID is generated with prefix "pathway".
ORGANISM	string. Organism property of the pathway. optional.
COMMENT	string. An optional comment

Value

Returns the biopax model with the added pathway.

Author(s)

fkramer

Examples

```
biopax = createBiopax(level=2)
biopax = addPhysicalEntity(biopax, class="protein", id="p_id1", NAME="protein1")
biopax = addPhysicalEntityParticipant(biopax, "p_id1", id="PEP_p_id1")
biopax = addPhysicalEntity(biopax, class="protein", id="p_id2", NAME="protein2")
biopax = addPhysicalEntityParticipant(biopax, "p_id2", id="PEP_p_id2")
biopax = addBiochemicalReaction(biopax, LEFT=c("PEP_p_id1"), RIGHT=c("PEP_p_id2"), id="biochem_id_1")
biopax = addPhysicalEntity(biopax, class="protein", id="p_id3", NAME="controllerProtein1")
```

```
biopax = addPhysicalEntityParticipant(biopax, "p_id3", id="PEP_p_id3")
biopax = addControl(biopax, CONTROL_TYPE="ACTIVATION", CONTROLLER="PEP_p_id3", CONTROLLED="biochem_id_1", id="c_
biopax = addPathway(biopax, NAME="mypathway1", PATHWAY_COMPONENTS=c("c_id1"), id="pw_id1")
biopax$dt
```

`addPathwayComponents` *This function adds pathway components to an existing pathway*

Description

This function adds pathway components to an existing pathway. Property PATHWAY-COMPONENTS are references to IDs of interaction/pathways/pathwaySteps (or subclasses of those)

Usage

```
addPathwayComponents(biopax, id, PATHWAY_COMPONENTS = c())
```

Arguments

`biopax`	A biopax model
`id`	string. ID for the pathway
`PATHWAY_COMPONENTS`	character vector. IDs of the pathway components. This must be IDs of instances of type interaction/pathway/pathwayStep (or their subclasses).

Value

Returns the biopax model with the pathway components added to the pathway

Author(s)

fkramer

Examples

```
biopax = createBiopax(level=2)
biopax = addPhysicalEntity(biopax, class="protein", id="p_id1", NAME="protein1")
biopax = addPhysicalEntityParticipant(biopax, "p_id1", id="PEP_p_id1")
biopax = addPhysicalEntity(biopax, class="protein", id="p_id2", NAME="protein2")
biopax = addPhysicalEntityParticipant(biopax, "p_id2", id="PEP_p_id2")
biopax = addBiochemicalReaction(biopax, LEFT=c("PEP_p_id1"), RIGHT=c("PEP_p_id2"), id="biochem_id_1")
biopax = addPhysicalEntity(biopax, class="protein", id="p_id3", NAME="controllerProtein1")
biopax = addPhysicalEntityParticipant(biopax, "p_id3", id="PEP_p_id3")
biopax = addControl(biopax, CONTROL_TYPE="ACTIVATION", CONTROLLER="PEP_p_id3", CONTROLLED="biochem_id_1", id="c_
biopax = addPathway(biopax, NAME="mypathway1", PATHWAY_COMPONENTS=c(), id="pw_id1")
biopax = addPathwayComponents(biopax, id="pw_id1", PATHWAY_COMPONENTS=c("c_id1"))
biopax$dt
```

addPhysicalEntity *This function adds a new physical entity.*

Description

This function adds a new physical entity of chosen class to the biopax model. This is a convenience function to add physical entities, internally the function addBiopaxInstance is called with properties NAME and ORGANISM set.

Usage

```
addPhysicalEntity(
  biopax,
  class = c("dna", "rna", "protein", "smallMolecule", "complex")[1],
  NAME,
  id = NULL,
  ORGANISM = NULL,
  COMMENT = NULL
)
```

Arguments

biopax	A biopax model
class	string. Class of the physical entity to add, choose from c("dna","rna","protein","smallMolecule","complex
NAME	string. Name of the new physical entity
id	string. ID for the physical entity. If NULL a new ID is generated with prefix "physicalEntity".
ORGANISM	string. Organism property of the molecule. optional.
COMMENT	string. An optional comment

Value

Returns the biopax model with the added physical entity.

Author(s)

fkramer

Examples

```
biopax = createBiopax(level=2)
biopax = addBiopaxInstance(biopax, class="protein", id="id1", properties=list(NAME="protein1",COMMENT="this is my
biopax$dt
biopax = addPhysicalEntity(biopax, class="protein", id="id2", NAME="protein2", COMMENT="This is a protein added us
biopax$dt
```

`addPhysicalEntityParticipant` *This function adds a new physical entity participant.*

Description

This function adds a new physical entity participant instance, which is a placeholder for physicalEntity class instances in interactions. This is a convenience function to add physicalEntityParticipant instances, internally the function addBiopaxInstance is called.

Usage

```
addPhysicalEntityParticipant(biopax, referencedPhysicalEntityID, id = NULL)
```

Arguments

`biopax`	A biopax model
`referencedPhysicalEntityID`	string. ID the new physicalEntity instance to reference here.
`id`	string. ID for the physical entity participant. If NULL a new ID is generated with prefix "physicalEntityParticipant".

Value

Returns the biopax model with the added physicalEntityParticipant.

Author(s)

fkramer

Examples

```
biopax = createBiopax(level=2)
biopax = addPhysicalEntity(biopax, class="protein", id="p_id1", NAME="protein1")
biopax = addPhysicalEntityParticipant(biopax, "p_id1", id="PEP_p_id1")
biopax = addPhysicalEntity(biopax, class="protein", id="p_id2", NAME="protein2")
biopax = addPhysicalEntityParticipant(biopax, "p_id2", id="PEP_p_id2")
biopax = addBiochemicalReaction(biopax, LEFT=c("PEP_p_id1"), RIGHT=c("PEP_p_id2"), id="biochem_id1")
biopax$dt
```

addPropertiesToBiopaxInstance *This function adds new properties to an existing biopax instance.*

Description

This function adds new properties to an existing biopax instance.

Usage

```
addPropertiesToBiopaxInstance(biopax, id, properties)
```

Arguments

biopax	A biopax model
id	string. ID of the instance
properties	named list of properties.

Value

Returns the supplied biopax model with new properties added to this instance.

Author(s)

Frank Kramer

Examples

```
biopax = createBiopax(level=2)
biopax = addBiopaxInstance(biopax, class="protein", id="id1", properties=list(NAME="protein1",SYNONYMS="p1"))
biopax$dt
biopax = addPropertiesToBiopaxInstance(biopax, id="id1", properties=list(COMMENT="this is my first protein!"))
biopax$dt
```

biopax *Biopax example data set*

Description

A dataset containing two regulatory pathways encoded in Biopax Level 2 and parsed in via readBiopax().

Another dataset containing pathways encoded in Biopax Level 2 and parsed in via readBiopax().

Format

An example biopax model parsed in via readBiopax.

An example biopax model parsed in via readBiopax.

Examples

```
data(biopaxexample)
biopax
data(biopaxLevel3Example)
biopax
```

---

calcGraphOverlap *This function calculates the overlap of 2 graphs*

Description

This function calculates the overlap of supplied graph1 with graph2. Layout and weights of graph1 are kept.

Usage

```
calcGraphOverlap(graph1, graph2)
```

Arguments

graph1	graphNEL
graph2	graphNEL

Value

Returns a list containing the compared graphs and edge- and node-wise overlap between them.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
pwid2 = "pid_p_100146_hespathway"
mygraph1 = pathway2RegulatoryGraph(biopax, pwid1)
mygraph2 = pathway2RegulatoryGraph(biopax, pwid2)
calcGraphOverlap(mygraph1,mygraph2)
```

checkValidity *This function checks the supplied biopax model for validity.*

Description

This function checks the supplied biopax model for validity, concerning classes, properties, etc. Not yet implemented. Called internally by writeBiopax.

Usage

```
checkValidity(biopax)
```

Arguments

biopax	A biopax model

Value

logical. Returns TRUE is the biopax model is valid Biopax Level 2, or FALSE otherwise.

Author(s)

Frank Kramer

CLASS_INHERITANCE_BP2 *CLASS_INHERITANCE_BP2*

Description

Class inheritance relationships in Biopax Level 2.

Usage

```
CLASS_INHERITANCE_BP2
```

Format

A data frame with 46 rows and 2 columns

Details

A data.frame listing all direct superclasses for every Biopax Level 2 class. The variables are as follows:

- class. Name of the class
- superclass. Name of the superclass

CLASS_INHERITANCE_BP3 *CLASS_INHERITANCE_BP3*

Description

Class inheritance relationships in Biopax Level 3.

Usage

```
CLASS_INHERITANCE_BP3
```

Format

A data frame with 46 rows and 2 columns

Details

A data.frame listing all direct superclasses for every Biopax Level 3 class. The variables are as follows:

- class. Name of the class
- superclass. Name of the superclass

NOT UPDATED TO BP3 yet!

CLASS_PROPERTIES_BP2 *CLASS_PROPERTIES_BP2*

Description

Class properties in Biopax Level 2.

Usage

```
CLASS_PROPERTIES_BP2
```

Format

A data frame with 106 rows and 4 columns

Details

A data.frame listing all direct properties for every Biopax Level 2 class. Together with CLASS_INHERITANCE_BP2 this allows to list all properties, including the inherited ones, of every class.

The variables are as follows:

- class. Name of the class
- property. Name of the superclass
- property_type.Type of the property, value or reference
- cardinality. Maximum allowed cardinality of a property. Many properties may only be singular.

`CLASS_PROPERTIES_BP3` *CLASS_PROPERTIES_BP3*

Description

Class properties in Biopax Level 3.

Usage

```
CLASS_PROPERTIES_BP3
```

Format

A data frame with 106 rows and 4 columns

Details

A data.frame listing all direct properties for every Biopax Level 3 class. Together with CLASS_INHERITANCE_BP3 this allows to list all properties, including the inherited ones, of every class.

The variables are as follows:

- class. Name of the class
- property. Name of the superclass
- property_type.Type of the property, value or reference
- cardinality. Maximum allowed cardinality of a property. Many properties may only be singular.

colorGraphNodes *This function colors the nodes of a graph.*

Description

This function colors nodes of a graph, usually this is used to color subgraphs or add a color hue correlating with the expression level or fold change to the molecules.

Usage

```
colorGraphNodes(graph1, nodes, values, colors = c("greenred", "yellowred"))
```

Arguments

graph1	graphNEL
nodes	vector of node names specifiying which nodes to color. must be same length as parameter foldChanges
values	vector of values indicating fold changes, gene expression values or similar. colors are mapped linearly over the range of these values
colors	string. either "greenred" or "yellowred", specifying which color gradient to use.

Value

Returns a graph with specified nodes colored according to the foldChanges

Author(s)

Frank Kramer

Examples

```
# load data and retrieve wnt pathway
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
mygraph1 = pathway2RegulatoryGraph(biopax, pwid1)
mygraph1 = layoutRegulatoryGraph(mygraph1)
# retrieve all nodes
nodes = nodes(mygraph1)
# random expression data for your nodes
values = rnorm(length(nodes), mean=6, sd=2)
# color nodes of the graph
mygraph1 = colorGraphNodes(mygraph1, nodes, values, colors="greenred")
# plot the now colored graph
plotRegulatoryGraph(mygraph1, layoutGraph=FALSE)
```

`combineNodes` *This function gracefully combines nodes of a regulatory graph.*

Description

This gracefully combines nodes from a regulatory graph. This is basically a wrapper for graph::combineNodes(nodes, graph, newName, collapseFunction=max). If there are duplicated edges for the nodes, the maximum edgeweight will be used for the new connection.

Usage

```
combineNodes(nodes, graph, newName)
```

Arguments

`nodes`	vector of node names specifiying which nodes to combine.
`graph`	graphNEL
`newName`	string. Name of the newly created node that will combine the specified nodes.

Value

Returns a graph with specified nodes removed.

Author(s)

Frank Kramer

Examples

```
# load data and retrieve wnt pathway
data(biopaxexample)
```

`createBiopax` *This function creates a new Biopax model from scratch*

Description

This function creates a new Biopax model from scratch. This is not necessary if you want to parse a BioPAX export from a file, please see: readBiopax. Returns a biopax model, which is a list with named elements:

df The data.frame representing the biopax in R

ns_rdf RDF Namespace

ns_owl OWL Namespace

ns_bp Biopax Namespace

file NULL

Usage

```
createBiopax(level = 3)
```

Arguments

level	integer. Specifies the BioPAX level.

Value

A biopax model

Author(s)

Frank Kramer

Examples

```
biopax = createBiopax(level=2)
```

DATABASE_BIOPAX *DATABASE_BIOPAX*

Description

Databases available for direct download via downloadBiopaxData

Usage

```
DATABASE_BIOPAX
```

Format

A data frame with 46 rows and 4 columns

Details

A data.frame listing all available databases which can be directly downloaded (Homo Sapiens only) via function downloadBiopaxData. The variables are as follows:

- database. Name of the database
- model. Name of the ontology model
- version. Biopax level
- link. Link to the direct download

diffGraphs *This function returns the different nodes and edges between graph1 and graph2.*

Description

This function returns the different nodes and edges between graph1 and graph2. Layout options of graph1 are kept. Coloring currently not implemented.

Usage

```
diffGraphs(graph1, graph2, colorNodes = TRUE, colors = c("#B3E2CD", "#FDCDAC"))
```

Arguments

graph1	graphNEL
graph2	graphNEL
colorNodes	logical
colors	character vector of colors. If colorNodes==TRUE these colors are used for graph1 and graph2 respectivley.

Value

Return the diff between the graphs.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
pwid2 = "pid_p_100146_hespathway"
mygraph1 = pathway2RegulatoryGraph(biopax, pwid1)
mygraph2 = pathway2RegulatoryGraph(biopax, pwid2)
plotRegulatoryGraph(diffGraphs(mygraph1,mygraph2))
```

`downloadBiopaxData` *This function downloads Biopax data from online databases*

Description

This function has an internal list of download links for some online databases. It will retrieve the selected model from the selected database using RCurl. The downloaded file is (if needed) unzipped and ready to be used as input for rBiopaxParser::readBiopax. This function requires package RCurl to run. You can easily skip this step by downloading the exported file yourself and continuing with readBiopax.

Usage

```
downloadBiopaxData(
  database = "NCI",
  model = c("pid", "biocarta", "reactome", "kegg"),
  outputfile = "",
  version = "biopax2"
)
```

Arguments

`database`	string. Select which database you want to download from. Currently only NCI links have been stored.
`model`	string. Select which model/file you want to download. Currently NCI versions of the Pathway Interaction Database, Biocarta, Reactome and KEGG are linked.
`outputfile`	string. The file name to save the downloaded data in. If left empty the URL file name will be used. The unzipped file name can be different from this. Check the screen output of gunzip.
`version`	string. Select which Biopax Version you want to download.

Value

none. Check output for the name of the unzipped biopax .owl file.

Author(s)

fkramer

Examples

```
## Not run: file = downloadBiopaxData("NCI", "biocarta", version = "biopax2")
## Not run: biopax = readBiopax(file)
## Not run: biopax
```

generateNewUniqueID *This function generates a new unique id for a biopax model*

Description

This function generates a new unique id for a biopax model. Pass it an startin g point like "pathway" or "protein" to get a niceer looking id.

Usage

```
generateNewUniqueID(biopax, id = "")
```

Arguments

biopax	A biopax model
id	string. This is used as a prefix for the id.

Value

Returns an unused unique ID.

Author(s)

fkramer

getClassProperties *This function returns the properties of the supplied biopax class.*

Description

This function returns the properties of the supplied biopax class. It always considers inhertance. Every class inhertis the properties of its super classes. A table listing all available properties and their cardinalities (for Biopax Level 2).

Usage

```
getClassProperties(classname, biopaxlevel = 3)
```

Arguments

classname	A string containing a class name
biopaxlevel	Numeric. Specifies the Biopax Level to use.

Value

Returns a data.frame containing the properties and cardinalities of the supplied class

Author(s)

Frank Kramer

Examples

```
getClassProperties("control")
```

---

getInstanceClass *This function returns the class name of the instance.*

Description

This function returns the class name of the instance.

Usage

```
getInstanceClass(biopax, id)
```

Arguments

biopax	A biopax model
id	string

Value

Returns the class name of the biopax instance.

Author(s)

fkramer

Examples

```
# load data
data(biopaxexample)
getInstanceClass(biopax, id="ex_m_100650")
```

getInstanceProperty *This function returns all properties of the specified type for an instance.*

Description

This function returns all properties of the specified type for an instance. By default this function returns the NAME property of an instance.

Usage

```
getInstanceProperty(
  biopax,
  id,
  property = "NAME",
  includeAllNames = TRUE,
  biopaxlevel = 3
)
```

Arguments

biopax	A biopax model
id	string
property	string.
includeAllNames	logical. Biopax Level 3 brought 2 new name properties: displayName and standardName. Per default this return all names of an instance. Disable if you only want the NAME property.
biopaxlevel	integer. Set the biopax level here if you supply a data.table directly.

Value

Returns a character vector with all properties of the selected type for this instance. Returns NULL if no property data is found.

Author(s)

fkramer

Examples

```
# load data
data(biopaxexample)
getInstanceProperty(biopax, id="ex_m_100650", property="NAME")
getInstanceProperty(biopax, id="ex_m_100650", property="ORGANISM")
getInstanceProperty(biopax, id="ex_m_100650", property="COMPONENTS")
```

`getNeighborhood` *This function returns the neighborhood of a physicalEntity*

Description

This function searches the supplied biopax for interactions that are connected to the molecule or within 'depth' number of steps from it.

Usage

```
getNeighborhood(biopax, id, depth = 1, onlyInPathways = c(), biopaxlevel = 3)
```

Arguments

`biopax`	A biopax model
`id`	string. ID of a physicalEntity (dna, rna, protein, complex, smallMolecule)
`depth`	integer. Search depth, this specifies how far out from the specified molecule the neighborhood should be streched.
`onlyInPathways`	character vector of pathway IDs. Search only in these pathways for neighbors.
`biopaxlevel`	integer. Set the biopax level here if you supply a data.table directly.

Value

Returns ids of interactions within 'depth' number of steps of the specified physicalEntity

Author(s)

fkramer

`getParticipants` *This function is used internally by pathway2Graph to obtain physical entities participating in an interaction.*

Description

This function is used internally by pathway2Graph to obtain physical entities participating in an interaction.

Usage

```
getParticipants(
  pwComponentList,
  instance,
  biopaxlevel,
  splitComplexMolecules = FALSE,
  useIDasNodenames = TRUE
)
```

Arguments

pwComponentList	List of pathway components
instance	Biopax instance id
biopaxlevel	integer. Set the biopax level here if you supply a data.table directly.
splitComplexMolecules	logical. If TRUE complexes are split up into their components and the annotation of the components is added.
useIDasNodenames	logical. If TRUE nodes of the graph are named by their molecule IDs instead of using the NAME property. This can help with badly annotated/formatted databases.

Author(s)

Nirupama Benis

---

getReferencedIDs *This function returns a vector of ids of all instances referenced by the specified instance.*

Description

This function takes an id and a biopax model as input. The id of every instance that is referenced is returned. If recursive == TRUE this function recurses through all referenced IDs of the referenced instances and so on. "onlyFollowProperties" limits the recursivness to only certain properties, for example follow only complexes or physicalEntities.

Usage

```
getReferencedIDs(biopax, id, recursive = TRUE, onlyFollowProperties = c())
```

Arguments

biopax	A biopax model OR a compatible data.table
id	string. ID of the instance
recursive	logical
onlyFollowProperties	character vector

Value

Returns a character vector of IDs referenced by the supplied id in the supplied biopax model.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
listComplexComponents(biopax, id="ex_m_100650")
getReferencedIDs(biopax, id="ex_m_100650", recursive=FALSE)
getReferencedIDs(biopax, id="ex_m_100650", recursive=TRUE)
```

---

getReferencingIDs *This function returns a vector of ids of all instances that reference the supplied id.*

Description

This function takes an id and a biopax model as input. The id of every instance that references the supplied id is returned. If recursive == TRUE this function recurses through all referencing IDs of the referencing instances and so on. "onlyFollowProperties" limits the recursivness to only certain properties, for example follow only complexes or physicalEntities.

Usage

```
getReferencingIDs(biopax, id, recursive = TRUE, onlyFollowProperties = c())
```

Arguments

biopax	A biopax model
id	string. ID of the instance
recursive	logical
onlyFollowProperties	character vector

Value

Returns a character vector of IDs referencing the supplied id in the supplied biopax model.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
listComplexComponents(biopax, id="ex_m_100650")
getReferencingIDs(biopax, id="ex_m_100650", recursive=FALSE)
getReferencingIDs(biopax, id="ex_m_100650", recursive=TRUE)
```

getSubClasses *This function returns the subclasses of the supplied biopax class.*

Description

This function returns the subclasses of the supplied biopax class.

Usage

```
getSubClasses(classname, biopaxlevel = 3)
```

Arguments

classname	A string containing a class name
biopaxlevel	Numeric. Specifies the Biopax Level to use.

Value

Returns character vector containing the subclasses of the supplied class

Author(s)

Frank Kramer

Examples

```
getSubClasses("control")
```

getSuperClasses *This function returns the superclasses of the supplied biopax class.*

Description

This function returns the superclasses of the supplied biopax class.

Usage

```
getSuperClasses(classname, biopaxlevel = 3)
```

Arguments

classname	A string containing a class name
biopaxlevel	Numeric. Specifies the Biopax Level to use.

Value

Returns character vector containing the superclasses of the supplied class

Author(s)

Frank Kramer

Examples

```
getSuperClasses("control")
```

`getXrefAnnotations` *This function returns the annotations of the supplied instances.*

Description

This function returns the annotations of the supplied IDs in a data.table.

Usage

```
getXrefAnnotations(
  biopax,
  id,
  splitComplexes = FALSE,
  followPhysicalEntityParticipants = TRUE,
  biopaxlevel = 3
)
```

Arguments

`biopax`	A biopax model
`id`	vector of strings. IDs of instances to get annotations
`splitComplexes`	logical. If TRUE complexes are split up into their components and the annotation of the components is added.
`followPhysicalEntityParticipants`	logical. If TRUE physicalEntityParticipants are resolved to their corresponding physicalEntities and their annotation is added.
`biopaxlevel`	integer. Set the biopax level here if you supply a data.table directly.

Value

Returns data.table with annotations

Author(s)

fkramer

Examples

```
 # load data
 data(biopaxexample)
# example of annotation for a protein:
getXrefAnnotations(biopax, id="ex_m_100647")
# no annotations for exactly the complex
getXrefAnnotations(biopax, id="ex_m_100650")
# split up the complex and get annotations for all the molecules involved
getXrefAnnotations(biopax, id="ex_m_100650", splitComplexes=TRUE)
```

`hasProperty` *Checks if instances in the biopax data.table have a given property*

Description

Checks if instances in the biopax data.table have a given property

Usage

```
hasProperty(df, property)
```

Arguments

`df`	A data.frame with biopax instances
`property`	A string containing the name of the property to check for

Value

Returns TRUE for every row in the data.frame with contains the supplied property. Logical vector with length corresponding to the number of rows in the data.frame.

Author(s)

Frank Kramer

Examples

```
 # load data
 data(biopaxexample)
```

`internal_checkArguments` *This function checks the supplied arguments if they abid to the given restrictions*

Description

This function checks the supplied arguments if they abid to the given restrictions

Usage

```
internal_checkArguments(
  args = c(),
  allowedValues = list(),
  allowNULL = FALSE,
  allowNA = FALSE,
  allowEmptyString = TRUE,
  allowInf = TRUE
)
```

Arguments

`args`	The vector of arguments to check
`allowedValues`	A named list of values the argument of a this name is allowed to have
`allowNULL`	Logical, allow NULL or not
`allowNA`	Logical, allow NA or not
`allowEmptyString`	Logical, allow empty strings or not
`allowInf`	Logical, allow values of +/- infinity or not

Value

Returns 1 if all checks completed successfully, returns error message otherwise.

Author(s)

Frank Kramer

`internal_generateXMLfromBiopax` *This function generates the xmlTree from the supplied biopax model.*

Description

This function is used internally by writeBiopax. It can also be called directly with a fitting dataframe in list(df=data.frame()), but this will probably break things.

Usage

```
internal_generateXMLfromBiopax(biopax, namespaces = namespaces, verbose = TRUE)
```

Arguments

`biopax`	A biopax model
`namespaces`	A list of namespaces to use for the generated XML/RDF file
`verbose`	logical

Value

Returns the xmlTree generated from the supplied biopax model.

Author(s)

Frank Kramer

`internal_getBiopaxModelAsDataFrame` *This internal function parses the Biopax XML of the supplied biopax model and returns it in the data.frame format.*

Description

This internal function parses the Biopax XML of the supplied biopax model and returns it in the data.frame format.

Usage

```
internal_getBiopaxModelAsDataFrame(biopax, biopaxxml, verbose = TRUE)
```

Arguments

`biopax`	A biopax object
`biopaxxml`	Biopax XML file read in. See parseBiopax
`verbose`	logical

Value

Returns the parsed biopax model in the internal data.frame format.

Author(s)

Frank Kramer

`internal_NrOfXMLNodes` *This function is an internal function to count the Number of nodes and child nodes of an XMLNode.*

Description

This function is an internal function to count the Number of nodes and child nodes of an XMLNode.

Usage

```
internal_NrOfXMLNodes(myXMLNode)
```

Arguments

`myXMLNode`	XMLNode to analyze

Value

This function returns the number of Nodes and child Nodes an XMLNode has.

Author(s)

Frank Kramer

`internal_propertyListToDF` *Internal function to build a data.frame from the list of properties for a new instance*

Description

Internal function to build a data.frame from the list of properties for a new instance

Usage

```
internal_propertyListToDF(
  class,
  id,
  properties,
  namespace_rdf = "rdf",
  biopaxlevel = 2
)
```

Arguments

class	string. Class name
id	string. ID of the instance
properties	named list of properties.
namespace_rdf	string. This defines the rdf namespace to use.
biopaxlevel	integer. This sets the version of BioPAX to generate, level 2 and level 3 are supported at the moment.

Value

Returns a data.frame with the new properties for the given instance

Author(s)

Frank Kramer

---

internal_resolvePhysicalEntityParticipant

This function resolves physicalEntityParticipantIDs to their corresponding physicalEntityIDs

---

Description

This function resolves physicalEntityParticipantIDs to their corresponding physicalEntityIDs. Every physicalEntityParticipant corresponds exactly to one physicalEntity.

Usage

```
internal_resolvePhysicalEntityParticipant(biopax, physicalEntityId)
```

Arguments

biopax	A biopax model
physicalEntityId	string. IDs of physicalEntityParticipants to be resolved

Value

Returns ids of physicalEntity corresponding to the specified physicalEntityParticipantIDs

Author(s)

fkramer

`internal_XMLInstance2DF` *This function is an internal function that parses a Biopax XMLNode.*

Description

This function is an internal function that parses a Biopax XMLNode. Do not call it manually.

Usage

```
internal_XMLInstance2DF(myXMLNode, namespace_rdf, ret, rowcount)
```

Arguments

`myXMLNode`	XMLNode
`namespace_rdf`	String specifying the namespace to use for rdf:resource and rdf:datatype
`ret`	data.table object contaning the already parsed data to attach this instance to
`rowcount`	Numeric specifying the row at which further parsed data is inserted into the data.table

Value

Returns a list contianing the new rowcount and the instance id of the added instance

Author(s)

Frank Kramer

`intersectGraphs` *This function returns a graph computed by the insection of supplied graph1 and graph2.*

Description

This function returns a graph computed by the insection of supplied graph1 and graph2. Layout and weights of graph1 are kept.

Usage

```
intersectGraphs(graph1, graph2)
```

Arguments

`graph1`	graphNEL
`graph2`	graphNEL

Value

Returns the intersection of graph1 and graph2.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
pwid2 = "pid_p_100146_hespathway"
mygraph1 = pathway2RegulatoryGraph(biopax, pwid1)
mygraph2 = pathway2RegulatoryGraph(biopax, pwid2)
plotRegulatoryGraph(intersectGraphs(mygraph1,mygraph2))
```

`isOfClass` *Checks if instances in the biopax data.table are of the given class*

Description

This function checks if instances in the supplied biopax data.table are of a given class. If considerInheritance is set to TRUE it also checks if instances are of a given class or any of its inherited classes.

Usage

```
isOfClass(df, class, considerInheritance = FALSE, biopaxlevel = 2)
```

Arguments

df	A data.frame with biopax instances
class	A string containing the class name to check for
considerInheritance	Logical value indicating wether to consider inheritance or not
biopaxlevel	Numeric. Specifies the Biopax Level to use.

Value

Returns TRUE for every row in the data.frame which is of the supplied class

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
```

---

isOfNamespace *Check if a classname is preceeded by a certain namespace tag like in "namespace:classname"*

Description

This function checks if the supplied input string starts with a supplied namespace tag

Usage

```
isOfNamespace(classname, namespace = "bp")
```

Arguments

classname	A string containing the classname to check
namespace	A string giving the namespace to check for

Value

This function returns TRUE if the supplied classname string is preceeded with the supplied namespace string, and FALSE if not.

Author(s)

Frank Kramer

isURL — *Check if a string is an URL, preceeded by "http:"*

Description

This function checks if the supplied input string starts with "http:"

Usage

```
isURL(string)
```

Arguments

string	A string containing the classname to check

Value

This function returns TRUE if the supplied classname string starts with "http:", and FALSE if not.

Author(s)

Frank Kramer

layoutRegulatoryGraph — *This function generates a (more or less) beautiful layout for a regulatory graph.*

Description

This function generates a (more or less) beautiful layout for a regulatory graph. Call this after you generated a graph with pathway2RegulatoryGraph. Since beauty is always in the eye of the beholder consider this a starting point for making your graphs even nicer. Rgraphviz with dot layout is used. Edges are green/red with normal/tee arrowheads for activations/inhibitions. If you want to specifically paint subgraphs in different colors use lists of vectors with node names for parameter subgraphs and vector of color names for subgraphs.color for your choice of color. The output can be further tweaked by setting layout options using nodeRenderInfo(mygraph) <- list() ... See the Rgraphviz and Graphviz documentations.

Usage

```
layoutRegulatoryGraph(
  mygraph,
  label = "",
  node.fixedsize = FALSE,
  edge.weights = c("green", "black", "red"),
  edge.arrowheads = c("normal", "tee"),
```

```
  subgraphs = list(),
  subgraphs.colors = c("#B3E2CD", "#FDCDAC", "#F4CAE4", "#E6F5C9", "#FFF2AE")
)
```

Arguments

mygraph	graphNEL
label	Label of the graph
node.fixedsize	logical. If font size is fixed or variable in regards to the nodes.
edge.weights	vector. which colors to use for weighted edges
edge.arrowheads	vector. which arrowheads to use for weighted edges
subgraphs	A list of character vectors with node names defining the sub graphs.
subgraphs.colors	vector. which colors to use for subgraphs

Value

Returns the supplied graph in a layouted form with several parameters set for regulatory graph plotting.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
pwid2 = "pid_p_100146_hespathway"
mygraph = pathway2RegulatoryGraph(biopax, pwid1)
mygraph = layoutRegulatoryGraph(mygraph)
plotRegulatoryGraph(mygraph)
```

---

listComplexComponents *This function lists all components of a given complex.*

Description

This function returns a (unique) data.frame listing all component IDs, names and classes of the supplied complex.

Usage

```
listComplexComponents(biopax, id, returnIDonly = FALSE, biopaxlevel = 3)
```

Arguments

biopax	A biopax model
id	string. A complex ID
returnIDonly	logical. If TRUE only IDs of the components are returned. This saves tiem for looking up names for every single ID.
biopaxlevel	integer. Set the biopax level here if you supply a data.table directly.

Value

data.frame

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
listComplexComponents(biopax, id="ex_m_100650")
```

---

listInstances *Lists all instances that conform to the selection criteria.*

Description

Lists all instances that conform to the selection criteria. In contrast to selectInstances this function returns an easier to read list. This function returns an ordered data.table of class, id and name of the instances. Selection criteria are wether instances belong to a certain class or have the specified id or name. Setting a criteria to NULL ignores this criteria. If includeSubClasses is set to TRUE the class criteria is broadened to include all classes that inherit from the given class, e.g. if class="control" and includeSubClasses=TRUE the function will select catalyses and modulations too, since they are a subclass of class control.

Usage

```
listInstances(
  biopax,
  id = NULL,
  class = NULL,
  name = NULL,
  includeSubClasses = FALSE,
  returnIDonly = FALSE,
  biopaxlevel = 3
)
```

Arguments

biopax	A biopax model
id	string. ID of the instances to select
class	string. Class of the instances to select
name	string. Name of the instances to select
includeSubClasses	logical. If includeSubClasses is set to TRUE the class criteria is broadened to include all classes that inherit from the given class
returnIDonly	logical. If TRUE only IDs of the components are returned. This saves time for looking up names for every single ID.
biopaxlevel	integer. Set the biopax level here if you supply a data.table directly.

Value

Returns a data.frame containing all instances conforming to the given selection criteria. If returnIDonly=TRUE, only the selector for the internal data.table otherwise.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
# list all instances of class "protein"
listInstances(biopax, class="protein")
# list all instances of class "pathway"
listInstances(biopax, class="pathway")
# list all interaction including all subclasses of interactions
listInstances(biopax, class="interaction", includeSubClasses=TRUE)
```

---

listInteractionComponents

This function lists all components of a given interaction.

Description

This function returns a (unique) data.frame listing IDs, names and classes of all components of the supplied interaction.

Usage

```
listInteractionComponents(
  biopax,
  id,
  splitComplexes = TRUE,
  returnIDonly = FALSE,
  biopaxlevel = 3
)
```

Arguments

biopax	A biopax model
id	string. A complex ID
splitComplexes	logical. If TRUE complexes are split up into their components and the added to the listing.
returnIDonly	logical. If TRUE only IDs of the components are returned. This saves tiem for looking up names for every single ID.
biopaxlevel	integer. Set the biopax level here if you supply a data.table directly.

Value

data.frame

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
listInteractionComponents(biopax, id="ex_i_100036_activator_1")
```

---

listPathwayComponents *This function lists all pathway components of a given pathway.*

Description

This function returns a (unique) data.frame listing all component IDs, names and classes of the supplied pathway.

Usage

```
listPathwayComponents(
  biopax,
  id,
  includeSubPathways = TRUE,
  returnIDonly = FALSE,
  biopaxlevel = 3
)
```

Arguments

biopax	A biopax model
id	string. A pathway ID
includeSubPathways	logical. If TRUE the returned list will include subpathways and pathwaysteps as well.
returnIDonly	logical. If TRUE only IDs of the components are returned. This saves tiem for looking up names for every single ID.
biopaxlevel	integer. Set the biopax level here if you supply a data.table directly.

Value

data.frame

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
listPathwayComponents(biopax, id="pid_p_100002_wntpathway")
```

---

listPathways *This function returns a list of all pathway ids.*

Description

This function returns a vector of all pathway ids.

Usage

```
listPathways(biopax, biopaxlevel = 3)
```

Arguments

biopax	A biopax model
biopaxlevel	integer. Set the biopax level here if you supply a data.table directly.

Value

Returns a character vector containing the names of all pathways.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
listPathways(biopax)
```

---

mergePathways *This function merges two given pathways*

Description

This function merges two given pathways and appends it to the supplied biopax model. The user has to specify a new name for the pathways and can supply ID, ORGANISM and COMMENT properties for the new pathway. If no ID is supplied, a new unique ID is generated. If no organism property is supplied the organism property of the first pathway is re-used. If ORGANISM is NULL the property is not set. Optionally a comment can be added to the pathway.

Usage

```
mergePathways(
  biopax,
  pwid1,
  pwid2,
  NAME,
  id = NULL,
  ORGANISM = "",
  COMMENT = NULL
)
```

Arguments

biopax	A biopax model
pwid1	string. ID of first pathway to merge
pwid2	string. ID of second pathway to merge

NAME	string. Name of the new merged pathway
id	string. ID for the pathway. If NULL a new ID is generated with prefix "pathway".
ORGANISM	string. Organism property of the pathway. By default uses the same organism as the first supplied pathway. If NULL no organism property is set.
COMMENT	string. An optional comment

Value

A biopax model with the merged pathway added.

Author(s)

fkramer

---

pathway2AdjacancyMatrix

This function generates an adjacency matrix from the activations/inhibitions of a pathway in a biopax model. This function internally first calls pathway2RegulatoryGraph, then converts the regulatory graph to an adjacency matrix. See pathway2RegulatoryGraph for more details.

Description

This function generates an adjacency matrix from the activations/inhibitions of a pathway in a biopax model.

This function internally first calls pathway2RegulatoryGraph, then converts the regulatory graph to an adjacency matrix. See pathway2RegulatoryGraph for more details.

Usage

```
pathway2AdjacancyMatrix(
  biopax,
  pwid,
  expandSubpathways = TRUE,
  splitComplexMolecules = TRUE,
  useIDasNodenames = FALSE,
  verbose = TRUE
)
```

Arguments

biopax	A biopax model
pwid	string
expandSubpathways	logical. If TRUE subpathways are expanded into this graph, otherwise only this very pathway is used.
splitComplexMolecules	logical. If TRUE every complex is split up into its components. This leads to splitting a single node with name of the complex into several nodes with names of the components, these components all have identical edges.
useIDasNodenames	logical. If TRUE nodes of the graph are named by their molecule IDs instead of using the NAME property. This can help with badly annotated/formatted databases.
verbose	logical

Value

Returns the adjacency matrix representing the regulatory graph of the supplied pathway.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
pwid2 = "pid_p_100146_hespathway"
pathway2AdjacancyMatrix(biopax, pwid1)
```

---

pathway2Geneset *This function generates the gene set of a pathway. This function generates a gene set of all physicalEntity's of a pathway. First all interactions of the pathway are retrieved and all components of these interactions are then listed.*

Description

This function generates the gene set of a pathway.

This function generates a gene set of all physicalEntity's of a pathway. First all interactions of the pathway are retrieved and all components of these interactions are then listed.

Usage

```
pathway2Geneset(biopax, pwid, returnIDonly = FALSE, biopaxlevel = 3)
```

Arguments

biopax	A biopax model
pwid	string
returnIDonly	logical. If TRUE only IDs of the components are returned. This saves tiem for looking up names for every single ID.
biopaxlevel	integer. Set the biopax level here if you supply a data.table directly.

Value

Returns the gene set of the supplied pathway. Returns NULL if the pathway has no components.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
pathway2Geneset(biopax, pwid=pwid1)
```

---

pathway2Graph *This function generates a directed graph from all the interactions of a specified pathway in a biopax model. Edges with no direction are indicated by a 0 weight.*

Description

This function generates a directed graph from all the interactions of a specified pathway in a biopax model. Edges with no direction are indicated by a 0 weight.

Usage

```
pathway2Graph(
  biopax,
  pwid,
  expandSubpathways = TRUE,
  splitComplexMolecules = FALSE,
  useIDasNodenames = TRUE,
  verbose = FALSE,
  withDisconnectedParts = TRUE
)
```

Arguments

biopax	A biopax model
pwid	string
expandSubpathways	logical. If TRUE subpathways are expanded into this graph, otherwise only this very pathway is used.
splitComplexMolecules	logical. If TRUE every complex is split up into its components. This leads to splitting a single node with name of the complex into several nodes with names of the components, these components all have identical edges. Default value is FALSE
useIDasNodenames	logical. If TRUE nodes of the graph are named by their molecule IDs instead of using the NAME property. This can help with badly annotated/formatted databases.
verbose	logical
withDisconnectedParts	logical. If TRUE the pathway graph is returned as such, else only the largest connected component is given back

Value

Returns the a graph object of the specified pathway. Edges with no direction are indicated by a 0 weight.

Author(s)

Nirupama Benis

Examples

```
 # load data
data(biopaxLevel3Example) # location of the data
pwid <- "Pathway1019"
# build pathway using pathway2Graph
pathwayAsGraph <- pathway2Graph(biopax = biopaxLevel3Example, pwid = pwid, splitComplexMolecules = FALSE, useIDasN
pathwayAsGraph # should have 23 nodes, 24 edges
plotRegulatoryGraph(pathwayAsGraph)
# build pathway discarding the disconnected parts of the graph
pathwayAsGraph <- pathway2Graph(biopax = biopaxLevel3Example, pwid = pwid, splitComplexMolecules = FALSE, useIDasN
pathwayAsGraph # should have 10 nodes, 11 edges
plotRegulatoryGraph(pathwayAsGraph)
```

pathway2RegulatoryGraph

This function generates the regulatory graph from the activations/inhibitions of a pathway in a biopax model. This functions builds a graph from the pathway components of the supplied pathway. Only instances of class 'control' are considered, this leads a functinal graph with all edges either representing activations or inhibitions. No transports, no translocation, etc. If desired complexes can be split up into several nodes, this can sometimes lead to a more complex and cluttered graph. There can not be multiple edges between 2 nodes. Whenever duplicated edges are generated (especially by splitting up complexes) a warning is thrown.

Description

This function generates the regulatory graph from the activations/inhibitions of a pathway in a biopax model.

This functions builds a graph from the pathway components of the supplied pathway. Only instances of class 'control' are considered, this leads a functinal graph with all edges either representing activations or inhibitions. No transports, no translocation, etc. If desired complexes can be split up into several nodes, this can sometimes lead to a more complex and cluttered graph. There can not be multiple edges between 2 nodes. Whenever duplicated edges are generated (especially by splitting up complexes) a warning is thrown.

Usage

```
pathway2RegulatoryGraph(
  biopax,
  pwid,
  expandSubpathways = TRUE,
  splitComplexMolecules = TRUE,
  useIDasNodenames = FALSE,
  verbose = TRUE
)
```

Arguments

biopax	A biopax model
pwid	string
expandSubpathways	logical. If TRUE subpathways are expanded into this graph, otherwise only this very pathway is used.
splitComplexMolecules	logical. If TRUE every complex is split up into its components. This leads to splitting a single node with name of the complex into several nodes with names of the components, these components all have identical edges.

useIDasNodenames
: logical. If TRUE nodes of the graph are named by their molecule IDs instead of using the NAME property. This can help with badly annotated/formatted databases.

verbose
: logical

Value

Returns the representing the regulatory graph of the supplied pathway in a node-edge-list graph.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
pwid2 = "pid_p_100146_hespathway"
mygraph = pathway2RegulatoryGraph(biopax, pwid1)
plotRegulatoryGraph(mygraph)
```

---

plotRegulatoryGraph *This function layouts a regulatory graph and plots it using Rgraphviz.*

Description

This function takes a regulatory graph as generated by pathway2regulatoryGraph and plots it using standard layout options of layoutRegulatoryGraph. This function is a wrapper for layoutRegulatoryGraph with standard parameters. Subgraphs can be painted with different colors. This can be done by passing parameter subgraph a list of character vectors with node names.

Usage

```
plotRegulatoryGraph(mygraph, subgraphs = list(), layoutGraph = TRUE)
```

Arguments

mygraph
: graphNEL, regulatory graph

subgraphs
: list of character vectors with node names

layoutGraph
: logical. If FALSE the graph is not laid out again but send directly to Rgraphviz::renderGraph.

Value

none

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
pwid2 = "pid_p_100146_hespathway"
mygraph = pathway2RegulatoryGraph(biopax, pwid1)
plotRegulatoryGraph(mygraph)
```

`print.biopax` *Print a biopax object.*

Description

Print a biopax object.

Usage

```
## S3 method for class 'biopax'
print(x, ...)
```

Arguments

`x`	A `biopax` object to print.
`...`	Other arguments to be passed to `print`.

Examples

```
data(biopaxexample)
print(biopax)
```

`readBiopax` *This function reads in a Biopax .owl file*

Description

This function reads in a Biopax .owl file and generates the internal data.frame format used in this package. This function can take a while with really big Biopax files like NCIs Pathway Interaction Database or Reactome. In almost every case this is your starting point. Returns a biopax model, which is a list with named elements:

df The data.frame representing the biopax in R

ns_rdf RDF Namespace

ns_owl OWL Namespace

ns_bp Biopax Namespace

file File name

Usage

```
readBiopax(file, verbose = TRUE)
```

Arguments

file	string. File name
verbose	logical. Output messages about how parsing is going and so on.

Value

A biopax model

Author(s)

Frank Kramer

Examples

```
## Not run: biopax = readBiopax(file="biopaxmodel.owl")
## Not run: biopax
#'  # load data and retrieve wnt pathway
data(biopaxexample)
```

---

removeDisconnectedParts

This function is used internally by pathway2Graph to remove the smaller disconnected parts of the pathway graph.

Description

This function is used internally by pathway2Graph to remove the smaller disconnected parts of the pathway graph.

Usage

```
removeDisconnectedParts(mygraph)
```

Arguments

mygraph	a graph object

Author(s)

Nirupama Benis

---

removeInstance *This function removes an instance*

Description

This function removes an instance from an existing biopax model.

Usage

```
removeInstance(biopax, id)
```

Arguments

biopax	A biopax model
id	string. ID of the instance

Value

Returns the supplied biopax model with the instance removed from it.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
biopax2 = removeInstance(biopax, 1)
```

removeNodes *This function gracefully removes nodes from a regulatory graph.*

Description

This function gracefully removes nodes from a regulatory graph. If the node to be removed has both parent and child nodes, these are connected directly. The weight of the new direct edge is the product of multiplying the incomming and outgoing edge weights of the original node.

Usage

```
removeNodes(graph, nodes)
```

Arguments

graph	graphNEL
nodes	vector of node names specifiying which nodes to remove.

Value

Returns a graph with specified nodes removed.

Author(s)

Frank Kramer

Examples

```
# load data and retrieve wnt pathway
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
mygraph1 = pathway2RegulatoryGraph(biopax, pwid1)
mygraph1 = layoutRegulatoryGraph(mygraph1)
# retrieve all nodes
nodes = nodes(mygraph1)
# random expression data for your nodes
values = rnorm(length(nodes), mean=6, sd=2)
# color nodes of the graph
mygraph1 = colorGraphNodes(mygraph1, nodes, values, colors="greenred")
# plot the now colored graph
plotRegulatoryGraph(mygraph1, layoutGraph=FALSE)
```

`removeProperties` *This function removes a property*

Description

This function removes a property fram an existing biopax instance.

Usage

```
removeProperties(biopax, id, properties)
```

Arguments

`biopax`	A biopax model
`id`	string. ID of the instance
`properties`	character vector. listing the properties to remove.

Value

Returns the supplied biopax model with properties removed from this instance.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
biopax2 = removeProperties(biopax, 1, "name")
```

`selectInstances` *Returns all instances that conform to the selection criteria.*

Description

Returns all instances that conform to the selection criteria. This function returns a subset of the internal data.table of the biopax object. Selection criteria are wether instances belong to a certain class or have the specified id, property or name. Setting a criteria to NULL ignores this criteria. If returnValues is set to FALSE only the selector (a logical vector with length of the internal data.table) is returned, otherwise the selected data is returned. If includeSubClasses is set to TRUE the class criteria is broadened to include all classes that inherit from the given class, e.g. if class="control" and includeSubClasses=TRUE the function will select catalyses and modulations too, since they are a subclass of class control. If includeReferencedInstances is set to TRUE all instances that are being referenced by the selected instances are being selected too. The parameter works recursively, this means for example that a selected pathway and all it's interactions, complexes, molecules and annotations are returned if this parameter is set to true. This parameter is especially helpful if you want to migrate or merge knowledge from different data bases.

Usage

```
selectInstances(
  biopax,
  id = NULL,
  class = NULL,
  property = NULL,
  name = NULL,
  returnValues = TRUE,
  includeSubClasses = FALSE,
  includeReferencedInstances = FALSE,
  returnCopy = TRUE,
  biopaxlevel = 3
)
```

Arguments

biopax	A biopax model or a compatible internal data.table
id	string. ID of the instances to select
class	string. Class of the instances to select
property	string. Return only this property of the instances
name	string. Name of the instances to select
returnValues	logical. If returnValues is set to FALSE only the selector (a logical vector with length of the internal data.table) is returned, otherwise the selected data is returned
includeSubClasses	logical. If includeSubClasses is set to TRUE the class criteria is broadened to include all classes that inherit from the given class
includeReferencedInstances	logical. If includeReferencedInstances is set to TRUE all instances that are being referenced by the selected instances are being selected too
returnCopy	logical. Defaults to TRUE. If TRUE a copy of the internal data.table is returned. If FALSE data is returned by reference. Set to FALSE to increase speed when only ever reading data. Make sure you understand the implications of using this! See vignette of data.table package.
biopaxlevel	integer. Set the biopax level here if you supply a data.table directly.

Value

Returns a data.table containing all instances conforming to the given selection criteria if returnValues=TRUE, only the selector for the internal data.table otherwise.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
# select the subset of the internal data.table that belongs to class "protein"
selectInstances(biopax, class="protein")
# select the subset of the internal data.table that belongs to class "interaction"
selectInstances(biopax, class="interaction")
# select the subset of the internal data.table that belongs to class "interaction" or any of its sub classes, like c
selectInstances(biopax, class="interaction", includeSubClasses=TRUE)
# select the subset of the internal data.table that belongs to class "pathway" AND is a "NAME" property
selectInstances(biopax, class="pathway", property="NAME")
```

`splitComplex` *This functions splits up a complex into its components.*

Description

This function looks up the supplied Complex ID and returns the names of all its components.

Usage

```
splitComplex(
  biopax,
  complexid,
  recursive = TRUE,
  returnIDonly = FALSE,
  biopaxlevel = 3
)
```

Arguments

`biopax`	A biopax model
`complexid`	string ID of an complex
`recursive`	logical
`returnIDonly`	logical. If TRUE only IDs of the components are returned. This saves tiem for looking up names for every single ID.
`biopaxlevel`	integer. Set the biopax level here if you supply a data.table directly.

Value

Returns a character vector with the names of all subcomponents.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
selectInstances(biopax, id="ex_m_100650")
listInstances(biopax, id="ex_m_100650")
listComplexComponents(biopax, id="ex_m_100650")
splitComplex(biopax, complexid="ex_m_100650")
```

---

striphash *Strips a hash in front of a string*

Description

Strips a hash in front of a string

Usage

```
striphash(x)
```

Arguments

x	A string to be stripped off a preceeeding hash

Value

The supplied string with a hash "#" stripped off front.

Author(s)

Frank Kramer

---

stripns *Strips a namespace tag off a supplied classname string*

Description

Strips a namespace tag off a supplied classname string

Usage

```
stripns(classname)
```

Arguments

classname	A string containing a classname preceeded by a namespace tag

Value

The classname with the namespace tag stripped off it.

Author(s)

Frank Kramer

`transitiveClosure` *This function generates the transitive closure of the supplied graph.*

Description

This function generates the transitive closure of the supplied graph. In short: if A->B->C then an edge A->C is added. Edge weights are conserved if possible (in a hopefully smart way). This is a simple convenience wrapper for the RBGL function transitive.closure.

Usage

```
transitiveClosure(mygraph)
```

Arguments

`mygraph`	graphNEL

Value

Returns the transitive closure of the supplied graph.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
pwid2 = "pid_p_100146_hespathway"
mygraph = pathway2RegulatoryGraph(biopax, pwid1)
tc = transitiveClosure(mygraph)
```

transitiveReduction *This function generates the transitive reduction of the supplied graph.*

Description

This function is deprecated due to nem dropping out of Bioconductor in BioC 4.0. This function generates the transitive reduction of the supplied graph. In short: if A->B->C AND A->C then edge A->C is removed. This is a simple convenience wrapper for the NEM function transitive.reduction. Be aware of implications on the edge weights!

Usage

```
transitiveReduction(mygraph)
```

Arguments

mygraph	graphNEL

Value

Returns the transitive reduction of the supplied graph.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
pwid2 = "pid_p_100146_hespathway"
mygraph = pathway2RegulatoryGraph(biopax, pwid1)
tr = transitiveReduction(mygraph)
```

unfactorize *Replace factors/levels in a data.frame and use plain strings instead*

Description

This function takes a data.frame as argument and returns it with strings instead of factors.

Usage

```
unfactorize(df)
```

Arguments

df	any data.frame with factor levels in at least one column

Value

The data.frame is returned using strings instead of factors.

Author(s)

Frank Kramer

---

uniteGraphs *This function unites two graphs.*

Description

This function unites the two supplied graphs. Layout parameters from graph1 are used. If colorNodes==TRUE the returned graph has different colors for overlapping nodes and nodes individual for each graph.

Usage

```
uniteGraphs(
  graph1,
  graph2,
  colorNodes = TRUE,
  colors = c("#B3E2CD", "#FDCDAC", "#F4CAE4")
)
```

Arguments

graph1	graphNEL
graph2	graphNEL
colorNodes	logical
colors	colors character vector of colors. If colorNodes==TRUE these colors are used for graph1 and graph2 respectivley.

Value

Return a graph generated by uniting the two supplied graphs

Author(s)

Frank Kramer

Examples

```
# load data
data(biopaxexample)
pwid1 = "pid_p_100002_wntpathway"
pwid2 = "pid_p_100146_hespathway"
mygraph1 = pathway2RegulatoryGraph(biopax, pwid1)
mygraph2 = pathway2RegulatoryGraph(biopax, pwid2)
plotRegulatoryGraph(uniteGraphs(mygraph1,mygraph2))
```

`writeBiopax` *This function writes out a biopax model.*

Description

This function writes out a biopax model, as generated by readBiopax, to either a file or returns the xmlTree if file is omitted.

Usage

```
writeBiopax(
  biopax,
  file = "",
  verbose = TRUE,
  overwrite = FALSE,
  namespaces = list(rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#", bp =
    "http://www.biopax.org/release/biopax-level2.owl#", rdfs =
   "http://www.w3.org/2000/01/rdf-schema#", owl = "http://www.w3.org/2002/07/owl#", xsd
    = "http://www.w3.org/2001/XMLSchema#")
)
```

Arguments

`biopax`	A biopax model as generated by readBiopax
`file`	A string giving a file name.
`verbose`	logical
`overwrite`	logical, if TRUE an already existing file will be overwritten, otherwise an error is thrown
`namespaces`	A list of namespaces to use for the generated XML/RDF file

Value

Returns the xmlTree object generated from the biopax model. If a filename is supplied the XML is written to this file.

Author(s)

Frank Kramer

Examples

```
# load data
data(biopax2example)
## Not run: writeBiopax(biopax, file="mybiopax.owl")
```

Index

∗ **datasets**
biopax, 13
CLASS_INHERITANCE_BP2, 15
CLASS_INHERITANCE_BP3, 16
CLASS_PROPERTIES_BP2, 16
CLASS_PROPERTIES_BP3, 17
DATABASE_BIOPAX, 20
∗ **package**
rBiopaxParser-package, 3

addBiochemicalReaction, 4
addBiopaxInstance, 5
addBiopaxInstances, 6
addControl, 6
addhash, 8
addns, 8
addPathway, 9
addPathwayComponents, 10
addPhysicalEntity, 11
addPhysicalEntityParticipant, 12
addPropertiesToBiopaxInstance, 13

biopax, 13
biopaxexample *(*biopax*)*, 13
biopaxLevel3Example *(*biopax*)*, 13

calcGraphOverlap, 14
checkValidity, 15
CLASS_INHERITANCE_BP2, 15
CLASS_INHERITANCE_BP3, 16
CLASS_PROPERTIES_BP2, 16
CLASS_PROPERTIES_BP3, 17
colorGraphNodes, 18
combineNodes, 19
createBiopax, 19

DATABASE_BIOPAX, 20
diffGraphs, 21
downloadBiopaxData, 22

generateNewUniqueID, 23
getClassProperties, 23
getInstanceClass, 24
getInstanceProperty, 25
getNeighborhood, 26
getParticipants, 26
getReferencedIDs, 27
getReferencingIDs, 28
getSubClasses, 29
getSuperClasses, 29
getXrefAnnotations, 30

hasProperty, 31

internal_checkArguments, 32
internal_generateXMLfromBiopax, 33
internal_getBiopaxModelAsDataFrame, 33
internal_NrOfXMLNodes, 34
internal_propertyListToDF, 34
internal_resolvePhysicalEntityParticipant, 35
internal_XMLInstance2DF, 36
intersectGraphs, 37
isOfClass, 37
isOfNamespace, 38
isURL, 39

layoutRegulatoryGraph, 39
listComplexComponents, 40
listInstances, 41
listInteractionComponents, 42
listPathwayComponents, 43
listPathways, 44

mergePathways, 45

pathway2AdjacancyMatrix, 46
pathway2Geneset, 47
pathway2Graph, 48
pathway2RegulatoryGraph, 50
plotRegulatoryGraph, 51
print.biopax, 52

rBiopaxParser *(*rBiopaxParser-package*)*, 3
rBiopaxParser-package, 3
readBiopax, 52
removeDisconnectedParts, 53
removeInstance, 54
removeNodes, 55
removeProperties, 56

selectInstances, 56
splitComplex, 58
striphash, 59
stripns, 59

transitiveClosure, 60
transitiveReduction, 61

unfactorize, 61
uniteGraphs, 62

writeBiopax, 63