Package: proteinDiscover (via r-universe)

August 20, 2024

Type Package

Imports magrittr, rlang, dplyr, purrr, lubridate, RSQLite, pool, stringr, XML, tidyr, tidyselect, bit64

Title ProteinDiscover

Version 0.11.0

Author Ben Bruyneel <benbruyneel@gmail.com>

Maintainer Ben Bruyneel <benbruyneel@gmail.com>

Description Provides an interface to the data contained in Proteome Discoverer (Thermo Scientific) results.

License GPL (>= 3)

Encoding UTF-8

URL https://github.com/BenBruyneel/proteinDiscover

LazyData true

RoxygenNote 7.3.1

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://benbruyneel.r-universe.dev

RemoteUrl https://github.com/BenBruyneel/proteinDiscover

RemoteRef HEAD

RemoteSha 04b6e22d4113cf4a6ec8fbe835bea39c109b50f6

Contents

`allNodesTable` *Helper function that takes the result from the `nodes` function, which is a named list of parameter tables (from processing or consensus workflow), and puts it all in a single table with the names of the nodes as an extra column*

Description

Helper function that takes the result from the `nodes` function, which is a named list of parameter tables (from processing or consensus workflow), and puts it all in a single table with the names of the nodes as an extra column

Usage

```
allNodesTable(nodesList)
```

Arguments

`nodesList`	named list of tables of workflow (node) parameters. Intended as input here is the output from the `nodes` function

Value

data.frame, a large table of all node parameters

`analysisDefinition` *function that gets the first element of the AnalysisDefinitionXML column from the AnalysisDefinition table in a .pdResult file*

Description

function that gets the first element of the AnalysisDefinitionXML column from the AnalysisDefinition table in a .pdResult file

Usage

```
analysisDefinition(db)
```

Arguments

`db`	database access 'handle' pointing to a .pdResult file

Value

a named tree like list that contains the info like file names, study factors, correction factors, etc etc

`blobLength` *attempts to determine the length (in bytes) of the individual elements of a blob-type column of a data.frame. It should (!) return an integer value of course (as all elements are supposed to have the same length). Also: if all elements of the column are NA, the the result will be NaN*

Description

attempts to determine the length (in bytes) of the individual elements of a blob-type column of a data.frame. It should (!) return an integer value of course (as all elements are supposed to have the same length). Also: if all elements of the column are NA, the the result will be NaN

Usage

```
blobLength(blobList)
```

Arguments

`blobList`	one column of a data.frame (as a list) of blob (raw) element type elements

Value

the length of the elements in the data.frame (or list) column. Again: this should be an integer

Note

meant for use in debugging problems

`calcAllIFIs` *Wrapper function that uses `tmt11Channels` to calculate the IFI's for a set of (knock out) protein channels*

Description

Wrapper function that uses `tmt11Channels` to calculate the IFI's for a set of (knock out) protein channels

Usage

```
calcAllIFIs(
  db,
  proteinsKnockedOut = knockOutProteins()$short[knockOutProteins()$knockout],
  accession = NA,
  groups = tmt11Channels(),
  joined = TRUE
)
```

Arguments

`db`	database access 'handle'
`proteinsKnockedOut`	character vector that specifies the (knock out) protein channels for which the IFI's are to be calculated
`accession`	single element character vector specifying the accession of the protein whose abundances are to be used for the IFI calculation
`groups`	usually either tmt10Channels() or tmt11Channels: data.frame that specifies which (abundance) column belongs to which knock out group
`joined`	defines the type of output: if TRUE then a single data.frame with all IFI's for all (knock out) proteins is generated. Otherwise a list of data.frame's is generated for all proteins separately

Value

a data.frame with two columns: one with the (short) name of the (selected) proteins and one with the calculated values (named IFI) or a list of data.frame's with the same structure

calcData *helper function to calculate a row-wise function (like mean, median etc) across a data.frame*

Description

helper function to calculate a row-wise function (like mean, median etc) across a data.frame

Usage

```
calcData(
  data,
  setNAZero = NA,
  removeNAs = FALSE,
  keepData = FALSE,
  calcName = "median",
  calcFunc = stats::median,
  ...
)
```

Arguments

`data`	the data.frame. Note that all rows and columns are used, so selection, filtering, etc should be done beforehand
`setNAZero`	default = NA, when NA this is ignored. Otherwise all cells containing NA will be set to the value of setNAZero. When removeNAs = TRUE, this parameter is ignored
`removeNAs`	default = FALSE, if TRUE all rows containing NA's will be removed via na.omit()
`keepData`	if TRUE, then the original data is returned also
`calcName`	name of the column with the calculated values in it
`calcFunc`	function to be applied row-wise across the data.frame
`...`	serves to pass on "extra" arguments on to the calcFunc function, eg na.rm = TRUE in case of calcFunc = mean

Value

a data.frame with the calculated values as the only column or with the calculated values as a mew column calcIFIs *function to calculate the IFI (interference free index) of a protein entry in the protein table of a pdResult files. Note this can only be calculated on the knockout proteins in the TKO control sample: see* tmt10Channels *or* tmt11Channels *for the eligible proteins*

Description

function to calculate the IFI (interference free index) of a protein entry in the protein table of a pdResult files. Note this can only be calculated on the knockout proteins in the TKO control sample: see tmt10Channels or tmt11Channels for the eligible proteins

Usage

```
calcIFIs(
  db,
  selected = "His4",
 accession = knockOutProteins()$Accession[knockOutProteins()$short == selected],
  columns = "Abundances",
  groups = tmt11Channels(),
  IFIName = "IFI",
  calcFunc = mean,
  calcName = "mean",
  na.rm = TRUE
)
```

Arguments

db	database access 'handle'
selected	(short) name of the selected protein
accession	uniprot accession code of the selected protein. If parameter "selected" is one of the short names in knockOutProteins then doesn't need to be specified. Note that the accession does not need to be one of the accessions of the knockout proteins
columns	usually this will be "Abundances". It allows the selection of the correct (raw) columns as they come out of dfTransformRaws(), eg Abunances_1, Abundances_2, etc
groups	usually either tmt10Channels() or tmt11Channels: data.frame that specifies which (abundance) column belongs to which knock out group. Note that the 'selected' argument should be in groups
IFIName	specifies the name to give to the calculated values, usually "IFI"
calcFunc	function to be applied row-wise across the data.frame. Used in the calculation of the IFI values. Default = mean
calcName	name of the column with the calculated values in it, used in the related function calcData()

`na.rm`	default = TRUE. This specifies that NA's should be removed when using eg mean, median, etc

Value

a data.frame with two columns: one with the (short) name of the (selected) protein and one with the calculated values (named IFI)

`columnSpecials` *Specials are not numeric or integer, but have chunks of a certain size All encountered in Proteome Discoverer are actually booleans with a value 0 (FALSE), 1 (TRUE) or NA*

Description

Specials are not numeric or integer, but have chunks of a certain size All encountered in Proteome Discoverer are actually booleans with a value 0 (FALSE), 1 (TRUE) or NA

Usage

```
columnSpecials()
```

Value

data.frame with columns 'names' and 'size'

Note

each chunk consists of two bytes, first one is logical (boolean): zero = FALSE, otherwise TRUE. Second byte = also logical: determines if value is NA (1) or not (0)

`createDiagrammeRString` *function to create a DiagrammeR string that can be used by DiagrammeR::grViz() to plot a visual representation of the workflow*

Description

function to create a DiagrammeR string that can be used by DiagrammeR::grViz() to plot a visual representation of the workflow

Usage

```
createDiagrammeRString(
  nodesTable,
  showBelow = TRUE,
  returnString = TRUE,
 hideDoubleParents = data.frame(name = c("Precursor Ions Quantifier", "Feature Mapper",
   "Reporter Ions Quantifier", "Protein Marker", "Peptide in Protein Annotation",
  "Modification Sites", "Peptide Isoform Grouper"), parent = c("last", "first", "last",
    "first", "first", "last", "first"))
)
```

Arguments

`nodesTable`	output from the nodeTable function. Columns that need to be present are node, name & parent
`showBelow`	boolean, default = TRUE. Set to FALSE when troubleshooting. Note that if set to FALSE, the parameter returnString will be ignored It is not recommended to depend on this parameter, as it will probably be removed in a newer version of the package
`returnString`	default = TRUE. Set to FALSE when troubleshooting. Note that the parameter showBelow makes it so that this parameter is ignored. It is not recommended to depend on this parameter, as it will probably be removed in a newer version of the package
`hideDoubleParents`	either NA (ignored) or a data.frame specifying what tp do in case of multiple parents. The data.frame should have the columns name and parent. The parent column should specify which parent to use ('first' or 'last') for connections

Value

character vector that can be passed on to DiagrammeR::grViz()

Note

during development it was noticed that some elements (nodes in the diagram) have more than one parent which is not seen in the proteome discoverer software of Thermo Scientific. The default data.frame 'corrects' known multiple parent nodes. If the parameter hideDoubleParents is set to NA, then the double parent connections are drawn.

an example of it's use: (workflowInfo(db))$nodeInfo$Consensus nodeTable() createDiagrammeR-String() grViz()

`dbClose` *Wrapper around pool::pooClose(): closes an open database (normally opened earlier via eg db_open())*

Description

Wrapper around pool::pooClose(): closes an open database (normally opened earlier via eg db_open())

Usage

```
dbClose(db)
```

Arguments

`db`	database access 'handle' to be closed

`dbGetAnnotatedProteins` *Function to get the UniqueSequenceID's for proteins which are in an protein annotation group. Essentially does the reverse of* `dbGetProteinAnnotationGroupIDs`. *The output of this function can serve as the input for* `dbGetProteins`

Description

Function to get the UniqueSequenceID's for proteins which are in an protein annotation group. Essentially does the reverse of `dbGetProteinAnnotationGroupIDs`. The output of this function can serve as the input for `dbGetProteins`

Usage

```
dbGetAnnotatedProteins(db, proteinAnnotationGroupIDs, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`proteinAnnotationGroupIDs`	the protein annotation group ID's for which to get the UniqueSequenceID's
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame or a character vector (SQL) `dbGetAnnotationGroups` *Function to get the info for (protein) annotation groups. Takes eg `dbGetProteinAnnotationGroupIDs` as input*

Description

Function to get the info for (protein) annotation groups. Takes eg `dbGetProteinAnnotationGroupIDs` as input

Usage

```
dbGetAnnotationGroups(
  db,
  proteinAnnotationGroupIDs = NA,
  columns = NA,
  SQL = FALSE
)
```

Arguments

`db`	database access 'handle'
`proteinAnnotationGroupIDs`	the protein annotation group ID's for which to get information
`columns`	allows the selection of columns to take from the table, default = NA (all columns)
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame or a character vector (SQL)

`dbGetAnnotationGroupsFiltered` *Get Group Annotation information from the table: AnnotationProteinGroups. This can be done via the GroupAnnotationAccession or via the description of an annotation. When using the Description it's possible to use the SQL 'like'*

Description

Get Group Annotation information from the table: AnnotationProteinGroups. This can be done via the GroupAnnotationAccession or via the description of an annotation. When using the Description it's possible to use the SQL 'like'

Usage

```
dbGetAnnotationGroupsFiltered(
  db,
  columns = NA,
  groupAnnotationAccession = NA,
  description = NA,
  UpperCase = FALSE,
  LowerCase = FALSE,
  like = FALSE,
  likePre = "%",
  likePost = "%",
  SQL = FALSE
)
```

Arguments

`db`	database access 'handle'
`columns`	allows the selection of columns to take from the table, default = NA (all columns)
`groupAnnotationAccession`	identification of the annotation, usually something like GO:.... (gene ontology) or pF.... (protein family). Note that when this argument is not NAm the arguments dealing with description etc are ignored
`description`	character vector specifying a word or sequence of word which is to be selected. If the 'like' argument is TRUE then it doesn't need to be exactly the same as the GroupAnnotationDescription field/column (in most cases the 'like' argument should be set to TRUE !)
`UpperCase`	if set to TRUE then BOTH description and the GroupAnnotationDescription field/column are entirely put to uppercase in the SQL used for the query. Note that if both UpperCase and LowerCase are TRUE, then UpperCase is used
`LowerCase`	if set to TRUE then BOTH description and the GroupAnnotationDescription field/column are entirely put to lowercase in the SQL used for the query.
`like`	if set to TRUE then the SQL 'LIKE' in stead of 'IN' is used to query the data. This only applies when the argument 'discription' is used. This is ignored when 'GroupAnnotationAccession' is used. If like = TRUE, then using eg 'locomotion' will result in the SQL query being: WHERE ... LIKE ' resulting table will give all rows, where the description part contains 'locomotion'. If like = FALSE, then only rows where the description exactly matches 'locomotion' will be selected. It's also possible to use the '_' (underscore) to make the LIKE function more or less specific. See eg SQL LIKE Operator for more info
`likePre`	default is ' 'description' argument to facilitate (partial) matching. It's better to set to " (empty string) when creating LIKE arguments directly via the 'description' argument
`likePost`	default is ' of the 'description' argument to facilitate (partial) matching. It's better to set to " (empty string) when creating LIKE arguments directly via the 'description' argument
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame or a character vector (SQL)

`dbGetConsensusIDs` *get the ConsensusID's from (a set of) PeptideGroupIDs*

Description

get the ConsensusID's from (a set of) PeptideGroupIDs

Usage

```
dbGetConsensusIDs(db, peptideGroupIDs, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`peptideGroupIDs`	the PeptideGroupIDs usually come from the TargetPeptideGroups Table. This can be in numeric or character vector format
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the TargetPeptideGroupsConsensusFeatures table or a character string specifying a SQL query

---

`dbGetConsensusTable` *get the Consensus Features table belonging to the ConsensusIDs*

Description

get the Consensus Features table belonging to the ConsensusIDs

Usage

```
dbGetConsensusTable(
  db,
  consensusIDs = NA,
  columns = NA,
  masterProtein = TRUE,
  sortorder = NA,
  SQL = FALSE
)
```

Arguments

`db`	database access 'handle'
`consensusIDs`	the PsmIDs to be retrieved. This can be in numeric or character vector format OR the output from the dbGetConsensusIDs function (a data.frame with column "ConsensusFeaturesId")
`columns`	allows the selection of columns to take from the table, default = NA (all columns)
`masterProtein`	use the IsMasterProtein column to be zero, default == TRUE. If more advanced filtering is needed, use db_getTable()
`sortorder`	allows for sorting of the selected columns, default = NA, (no sorting). Other valid values are a single character string ("ASC" or "DESC") or a character vector of the same length as the columnNames vector containing a series of "ASC" or "DESC"
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the peptide table or a character string specifying a SQL query

---

`dbGetMassSpectrumItems` *get the MassSpectrumItems info from (a set of) PeptideID's*

Description

get the MassSpectrumItems info from (a set of) PeptideID's

Usage

```
dbGetMassSpectrumItems(db, dbDetail = NA, peptideID, SQL = FALSE)
```

Arguments

`db`	database access 'handle' (to the .pdResult file)
`dbDetail`	database access 'handle' to the details file (.pdResultDetails). This is needed for at least Proteome Discover 3.1, since the "MassSpectrumItems" table is located in a different file than the e.g. the psm table. Note that if the 'SQL' parameter is set to TRUE, the function will only return the last SQL query (querying the .pdResultDetails table).
`peptideID`	the PeptideID's usually come from the PSMS table Table. This can be in numeric/character/data.frame format
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the MassSpectrumItems table or a character string specifying an SQL query

---

`dbGetModificationPeptideIDs`
Function to get the peptideID's 'belonging' to a modification site

---

Description

Function to get the peptideID's 'belonging' to a modification site

Usage

```
dbGetModificationPeptideIDs(db, modificationIDs, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`modificationIDs`	the modification site identifiers to get from the ModificationSites table. This should be the 'Id' field of a modifciation table row
`SQL`	allows the function to return the SQL query statement in stead of a data.frame#'

Value

a data.frame or a character vector (SQL)

`dbGetModificationsSitesIDs`
function to get the modificationSite ID's from (a set of) protein-UniqueID's

---

Description

function to get the modificationSite ID's from (a set of) proteinUniqueID's

Usage

```
dbGetModificationsSitesIDs(db, proteinUniqueIDs, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`proteinUniqueIDs`	the protein identifier for which the modificationSite ID's are to be fetched. This is a vector of one or more integer64 (package: bit64) values. In protein tables this is the UniqueSequenceUD column
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing the requested data from the TargetProteinsModificationSites table or a character string specifying an SQL query

Note

the data from modificationSitesUd's in the result can be used to query the ModificationSites table via `dbGetModificationsTable`

---

dbGetModificationsTable

function to get data from the ModificationSides table using the modificiationSiteId's

---

Description

function to get data from the ModificationSides table using the modificiationSiteId's

Usage

```
dbGetModificationsTable(
  db,
  modificatonSitesIDs,
  columns = NA,
  sortorder = NA,
  SQL = FALSE
)
```

Arguments

`db`	database access 'handle'
`modificatonSitesIDs`	the modification site identifiers to get from the ModificationSites table
`columns`	allows the selection of columns to take from the table, default = NA (all columns)
`sortorder`	allows for sorting of the selected columns, default = NA, (no sorting). Other valid values are a single character string ("ASC" or "DESC") or a character vector of the same length as the columnNames vector containing a series of "ASC" or "DESC"
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame or a character vector (SQL)

Note

the easiest way to get the modificationSitesIDs is via the `dbGetModificationsSitesIDs` function

`dbGetMSnSpectrumInfo` *get the MSnSpectrumInfo from (a set of) PeptideID's*

Description

get the MSnSpectrumInfo from (a set of) PeptideID's

Usage

```
dbGetMSnSpectrumInfo(db, peptideID, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`peptideID`	the PeptideID's usually come from the PSMS table Table. This can be in numeric/character/data.frame format
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the MSnSpectrumInfo table or a character string specifying an SQL query

`dbGetPeptideIDs` *get the peptideID's from (a set of) proteinGroupIDs*

Description

get the peptideID's from (a set of) proteinGroupIDs

Usage

```
dbGetPeptideIDs(db, proteinGroupIDs, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`proteinGroupIDs`	the proteinGroupIDs usually come from the TargetProtein Table. This can be in numeric or character vector format
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the TargetProteinGroupsTargetPeptideGroups table or a character string specifying an SQL query

Note

to get the proteinpeptidelink (table = "TargetProteinGroupsTargetPeptideGroups"). In goes "ProteinGroupID" from the table "TargetProteins" (Note: it's possible to use a c(„,) to get the result for a number of proteins at the same time). The result is a list of numbers which are the "TargetProteinGroupsProteinGroupID" in the "TargetPeptideGroups" table

---

`dbGetPeptideTable` *get the paptide table belonging defined by PeptideIDs ot proteinGroupIDs*

Description

get the paptide table belonging defined by PeptideIDs ot proteinGroupIDs

Usage

```
dbGetPeptideTable(
  db,
  peptideIDs = NA,
  proteinGroupIDs = NA,
  columns = NA,
  masterProtein = TRUE,
  sortorder = NA,
  SQL = FALSE
)
```

Arguments

`db`	database access 'handle'
`peptideIDs`	the peptideIDs to be retrieved. This can be in numeric or character vector format OR the output from the dbGetPeptideIDs function (a data.frame with column "TargetPeptideGroupsPeptideGroupID")
`proteinGroupIDs`	the proteinGroupIDs usually come from the TargetProtein Table. This can be in numeric or character vector format. Note: if this parameter is not NA, then peptideIDs will be ignored. This makes it possible to retrieve the peptides belonging to a protein w/o first having to retrieve toe Peptide ID's
`columns`	allows the selection of columns to take from the table, default = NA (all columns)
`masterProtein`	use the IsMasterProtein column to be zero, default == TRUE. If more advanced filtering is needed, use db_getTable()
`sortorder`	allows for sorting of the selected columns, default = NA, (no sorting). Other valid values are a single character string ("ASC" or "DESC") or a character vector of the same length as the columnNames vector containing a series of "ASC" or "DESC"
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the peptide table or a character string specifying a SQL query

---

`dbGetProteinAnnotationGroupIDs`
Function to get the functional group annotation group ID's for proteins. This function does essentially the reverse of `dbGetAnnotatedProteins`*. The output of this function can serve as the input for* `dbGetAnnotationGroups`

Description

Function to get the functional group annotation group ID's for proteins. This function does essentially the reverse of `dbGetAnnotatedProteins`. The output of this function can serve as the input for `dbGetAnnotationGroups`

Usage

```
dbGetProteinAnnotationGroupIDs(db, uniqueSequenceIDs, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`uniqueSequenceIDs`	the UniqueSequenceID's (unique protein identifier), usually coming from protein table
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame or a character vector (SQL)

`dbGetProteinFiltered` *A bit more advanced version of* `dbGetProteinTable` *which allows for filtering (via SQL). Note that filtering raw columns (BLOB's) will not work properly*

Description

A bit more advanced version of `dbGetProteinTable` which allows for filtering (via SQL). Note that filtering raw columns (BLOB's) will not work properly

Usage

```
dbGetProteinFiltered(
  db,
  columns = NA,
  masterProtein = FALSE,
  sortorder = NA,
  filtering = NA,
  SQL = FALSE
)
```

Arguments

`db`	database access 'handle'
`columns`	allows the selection of columns to take from the table, default = NA (all columns)
`masterProtein`	use the IsMasterProtein column to be zero, default == TRUE. If more advanced filtering is needed, use db_getTable() Note that if set to FALSE then no filtering is performed on the status of the IsMasterProtein column
`sortorder`	allows for sorting of the selected columns, default = NA, (no sorting). Other valid values are a single character string ("ASC" or "DESC") or a character vector of the same length as the columnNames vector containing a series of "ASC" or "DESC"
`filtering`	SQL statement to be used for filtering of the query. The IsMasterProtein column is already covered when masterProtein is set to TRUE
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the protein table or a character string specifying an SQL query

---

`dbGetProteinGroupIDs` *Retrieve the ProteinGroupID's of proteins via their UniqueSequenceID's*

Description

Retrieve the ProteinGroupID's of proteins via their UniqueSequenceID's

Usage

```
dbGetProteinGroupIDs(db, proteinUniqueIDs, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`proteinUniqueIDs`	the UniqueSequenceID's for which the proteinGroupID's are to be retrieved. Usually these UniqueSequenceID's will come from a protein table. Please note that a 'regular' bit64::as.integer64 vector may fail due to conversion issues. It is better to pass this type of vector as a character vector
`SQL`	allows the function to return the SQL query statement in stead of a data.frame#'

Value

a data.frame or a character vector (SQL)

Note

the output of this is meant to serve as input for the `dbGetProteinGroups` function

---

`dbGetProteinGroups` *Gets the ProteinGroup information from the TargetProteinGroups table*

Description

Gets the ProteinGroup information from the TargetProteinGroups table

Usage

```
dbGetProteinGroups(db, proteinGroupIDs, columns = NA, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`proteinGroupIDs`	specifies which protein groups to get, these values can come from eg the protein table
`columns`	character vector, specifies which columns to retrieve
`SQL`	allows the function to return the SQL query statement in stead of a data.frame#'

Value

a data.frame or a character vector (SQL)

`dbGetProteinIDs` *Function to get proteinUniqueID's from a (set of) protein groupID's (eg from a proteinGroup tables, or dbGetProteinGroupIDs). This allows for getting all proteins (also non-master proteins) which together make up a protein group. Normally only the master protein is shown in a protein table*

Description

Function to get proteinUniqueID's from a (set of) protein groupID's (eg from a proteinGroup tables, or dbGetProteinGroupIDs). This allows for getting all proteins (also non-master proteins) which together make up a protein group. Normally only the master protein is shown in a protein table

Usage

```
dbGetProteinIDs(db, proteinGroupIDs, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`proteinGroupIDs`	the protein group(s) for which the UniqueSequenceID's should be retrieved. This can also be a (collpased) character vector where the protein groups are separated by ';'
`SQL`	allows the function to return the SQL query statement in stead of a data.frame#'

Value

a data.frame or a character vector (SQL)#'

Note

every protein in the protein table has a ProteinGroupID & a UniqueSequenceID. The UniqueSequenceID is untique to the protein. A protein group may contain more than one protein (and thus also more than one UniqueSequenceID)

`dbGetProteins` *Function to get protein information from the TargetProteins table on the basis of their UniqueSequenceID*

Description

Function to get protein information from the TargetProteins table on the basis of their UniqueSequenceID

Usage

```
dbGetProteins(db, UniqueSequenceIDs, columns = NA, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`UniqueSequenceIDs`	character vector that specifies for which proteins to get info. Please note that in the 'TargetProteins' table the column 'UniqueSequenceID' is integer64 class. To prevent issues these values should be converted to character vector(s).
`columns`	character vector, specifies which columns to retrieve
`SQL`	allows the function to return the SQL query statement in stead of a data.frame#'

Value

a data.frame or a character vector (SQL)

`dbGetProteinTable` *get the protein table from a .pdResult file (essentially a wrapper around db_getTable())*

Description

get the protein table from a .pdResult file (essentially a wrapper around db_getTable())

Usage

```
dbGetProteinTable(
  db,
  columns = NA,
  masterProtein = TRUE,
  sortorder = NA,
  SQL = FALSE
)
```

Arguments

`db`	database access 'handle'
`columns`	allows the selection of columns to take from the table, default = NA (all columns)
`masterProtein`	use the IsMasterProtein column to be zero, default == TRUE. If more advanced filtering is needed, use db_getTable()
`sortorder`	allows for sorting of the selected columns, default = NA, (no sorting). Other valid values are a single character string ("ASC" or "DESC") or a character vector of the same length as the columnNames vector containing a series of "ASC" or "DESC"
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the protein table or a character string specifying a SQL query

---

`dbGetProteinUniqueSequenceIDs` *Function to retrieve the UniqueSequenceID's based on the accession field of the proteinTable. Essentially a wrapper for* `dbGetProteinFiltered`

Description

Function to retrieve the UniqueSequenceID's based on the accession field of the proteinTable. Essentially a wrapper for `dbGetProteinFiltered`

Usage

```
dbGetProteinUniqueSequenceIDs(db, accession = NA, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`accession`	accession(s) of the proteins
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame or a character vector (SQL)

`dbGetPsmIDs` *get the PsmID's from (a set of) PeptideGroupIDs*

Description

get the PsmID's from (a set of) PeptideGroupIDs

Usage

```
dbGetPsmIDs(db, peptideGroupIDs, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`peptideGroupIDs`	the PeptideGroupIDs usually come from the TargetPeptideGroups Table. This can be in numeric or character vector format
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the TargetPsmsTargetPeptideGroups table or a character string specifying an SQL query

---

`dbGetPsmTable` *get the PSM table belonging to the PsmIDs*

Description

get the PSM table belonging to the PsmIDs

Usage

```
dbGetPsmTable(
  db,
  psmIDs = NA,
  peptideGroupIDs = NA,
  columns = NA,
  masterProtein = TRUE,
  sortorder = NA,
  filtering = "MasterProteinAccessions IS NOT NULL",
  SQL = FALSE
)
```

Arguments

`db`	database access 'handle'
`psmIDs`	the PsmIDs to be retrieved. This can be in numeric or character vector format OR the output from the dbGetPsmIDs function (a data.frame with column "TargetPsmsPeptideID")
`peptideGroupIDs`	the PeptideGroupIDs usually come from the TargetPeptideGroups Table. This can be in numeric or character vector format Note: if this parameter is not NA, then psmIDs will be ignored. This makes it possible to retrieve the psm info belonging to a peptide w/o first having to retrieve toe psm ID's
`columns`	allows the selection of columns to take from the table, default = NA (all columns)
`masterProtein`	use the IsMasterProtein column to be zero, default == TRUE. If more advanced filtering is needed, use db_getTable()
`sortorder`	allows for sorting of the selected columns, default = NA, (no sorting). Other valid values are a single character string ("ASC" or "DESC") or a character vector of the same length as the columnNames vector containing a series of "ASC" or "DESC"
`filtering`	allows for " WHERE <expression>" additions to the SQL statement default = " " (no filtering). Note: always put a space (" ") before any statement. If NA then no filtering is applied. Note that filtering is only used when the argument PsmIDs is not NA
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the peptide table or a character string specifying a SQL query

---

dbGetQuanSpectrumIDs *get the SpectrumID's from (a set of) PeptideIDs*

Description

get the SpectrumID's from (a set of) PeptideIDs

Usage

```
dbGetQuanSpectrumIDs(db, peptideIDs, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`peptideIDs`	the PeptideIDs usually come from the TargetPsms Table. This can be in numeric or character vector format
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the TargetPsmsQuanSpectrumInfo table or a character string specifying an SQL query

---

dbGetQuanSpectrumInfoTable *get the QuanSpectrumInfo table belonging to the SpectrumID's*

Description

get the QuanSpectrumInfo table belonging to the SpectrumID's

Usage

```
dbGetQuanSpectrumInfoTable(
  db,
  spectrumIDs = NA,
  columns = NA,
  masterProtein = TRUE,
  sortorder = NA,
  SQL = FALSE
)
```

Arguments

`db`	database access 'handle'
`spectrumIDs`	the SpectrumID's to be retrieved. This can be in numeric or character vector format OR the output from the dbGetQuanSpectrumIDs function (a data.frame with column "QuanSpectrumInfoSpectrumID")
`columns`	allows the selection of columns to take from the table, default = NA (all columns)
`masterProtein`	use the IsMasterProtein column to be zero, default == TRUE. If more advanced filtering is needed, use db_getTable()
`sortorder`	allows for sorting of the selected columns, default = NA, (no sorting). Other valid values are a single character string ("ASC" or "DESC") or a character vector of the same length as the columnNames vector containing a series of "ASC" or "DESC"
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from the QuanSpectrumInfo table or a character string specifying a SQL query

---

`dbGetTable` *get a table from a .pdResult file*

Description

get a table from a .pdResult file

Usage

```
dbGetTable(
  db,
  tablename,
  columns = NA,
  filtering = " ",
  sortorder = NA,
  SQL = FALSE
)
```

Arguments

`db`	database access 'handle'
`tablename`	used to pass on the name of the table containing the data
`columns`	allows the selection of columns to take from the table, default = NA (all columns)
`filtering`	allows for " WHERE <expression>" additions to the SQL statement default = " " (no filtering). Note: always put a space (" ") before any statement

`sortorder`	allows for sorting of the selected columns, default = NA, (no sorting). Other valid value is a character character vector of columnNames to be used for sorting string (with "ASC" or "DESC" if needed)
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

a data.frame containing requested data from a database table or a character string specifying an SQL query

---

`dbOpen` *Wrapper around pool::dbPool(): opens a database*

Description

Wrapper around pool::dbPool(): opens a database

Usage

```
dbOpen(filename, drv = RSQLite::SQLite(), ...)
```

Arguments

`filename`	a character vector specifying the name and location of the database
`drv`	defines database connection type, default = RSQLite::SQLite()
`...`	to pass on additional parameters to pool::dbPool, exmples are host = "shiny-demo.csa7qlmguqrf.us-east-1.rds.amazonaws.com" username = "guest" password = "guest"

Value

database access 'handle'

Note

if no file with the name 'fileName' exists, then it will be created (but obviously it will be empty, so most further commands will fail)

if fileName == ":memory:" the database will be an in-memory database

---

`determineBlobTypes` *function that attempts to assign types and sizes to the blob type columns in a table. The result from this function can be used in the dfTransformRaws function*

Description

function that attempts to assign types and sizes to the blob type columns in a table. The result from this function can be used in the dfTransformRaws function

Usage

```
determineBlobTypes(
  theTable,
  minimumNumber = 1,
  numberOfGroups = minimumNumber,
  ratioNumberOfGroups = numberOfGroups - 1,
  blobDF = NA,
  specials = TRUE
)
```

Arguments

`theTable`	a data.frame with blob Columns (if no blobColumns are present, then NA is returned)
`minimumNumber`	this defines the minimum number of columns a blob/raw type column should be split into. In TMT10plex experiments, the minimumNumber will usually be 10, becauseyou have 10 channels/abundances
`numberOfGroups`	this defines how many 'groups' are present in the data. Taking Abundances as an example: Proteone Discoverer has both the original columns (say Abundances_1 through Abundances_2), but also columns where the abundances, that 'belong' together, are eg averaged or some other (statistical) measure is calculated over a number of columns. You may have eg 10 'Abundance channels' which are 5 samples total, each in duplo. This means that some columns in the resulting table will need to be split in 10 different columns (the original 'Abundances') while 'grouped' columns should be split into 5 different columns (eg the calculated means or variations of the 'abundances' columns). Note that although not enforced by the code, the numberOfGroups should always be equal or less than the minimumNumber parameter. Default value = minimumNumber
`ratioNumberOfGroups`	when ratios between groups are calculated we get columns (ratio columns) that need to be split into numberOfGroups - 1 (which is the efault value)
`blobDF`	essentially the result from either getBlobs; if NA then it will be generated by the getBlobs function with theTable as an argument
`specials`	default is TRUE, means that specials will be taken care of

Value

a data.frame with the name of the blob columns, their lengths, what (type) and minimumSize (number of variables in the blob)

Note

this function does not deal properly with specials, their types/ translations are resolved in a different way

there are two ways to see potential problems with the type assignments: the columns may contain NA values

---

`dfTransformRaws` *df_transform_raws(): converts raw columns in a data.frame to the correct data types*

Description

df_transform_raws(): converts raw columns in a data.frame to the correct data types

Usage

```
dfTransformRaws(
  df,
  blobDF = NA,
  minimumNumber = 1,
  numberOfGroups = minimumNumber,
  ratioNumberOfGroups = numberOfGroups - 1,
  specials = TRUE
)
```

Arguments

`df`	data.frame coming from a table from a Proteome Discoverer database (eg .pdResult files)
`blobDF`	must be data.frame with 4 columns: name (columnName), length (number of bytes per cell), what (type) & minimumSize (number of values in a cell) default = NA. If 'what' in the data.frame = NA, then the columnVector will not be converted, but returned as it is
`minimumNumber`	this defines the minimum number of columns a blob/raw type column should be split into. In TMT10plex experiments, the minimumNumber will usually be 10, becauseyou have 10 channels/abundances
`numberOfGroups`	this defines how many 'groups' are present in the data. Taking Abundances as an example: Proteone Discoverer has both the original columns (say Abundances_1 through Abundances_2), but also columns where the abundances, that 'belong' together, are eg averaged or some other (statistical) measure is calculated over a number of columns. You may have eg 10 'Abundance channels'

	which are 5 samples total, each in duplo. This means that some columns in the resulting table will need to be split in 10 different columns (the original 'Abundances') while 'grouped' columns should be split into 5 different columns (eg the calculated means or variations of the 'abundances' columns). Note that although not enforced by the code, the numberOfGroups should always be equal or less than the minimumNumber parameter. Default value = minimumNumber
`ratioNumberOfGroups`	when ratios between groups are calculated we get columns (ratio columns) that need to be split into numberOfGroups - 1 (which is the efault value)
`specials`	default is TRUE, means that specials will be taken care of

Value

data.frame with all raw vector ('blob') columns converted to more more regular data types

Note

the tables/data.frame's coming from a Proteome Discoverer database (eg .pdResult files) have columns of the type raw vecotr (blob). These can be converted automatically or semi-automatically by this function

If there are no raw vector columns, then this function has no use and may even trigger errors/warnings

This function can only do integer & numeric blob columns (and the specials) at the moment

some raw vector columns are actually two (or possibly more) columns in one. In those cases each element/cell of the column is two (or more) values. This function splits these columns into two seperate ones.

`df_replace` *function that replaces (parts of) strings in a data.frame according to a provided table of replacements*

Description

function that replaces (parts of) strings in a data.frame according to a provided table of replacements

Usage

```
df_replace(df, str_replacements = replacementStrings())
```

Arguments

`df`	data.frame that needs to have strings replaced. Each cell is processed with str_replace_all from the stringr package for all elements of the str_replacements data.frame
`str_replacements`	data.frame defining the replacements, see replacementStrings for more information

Value

the data.frame with (parts of) strings replaced if present

Note

this function can be called just before passing a data.frame over to eg kableExtra::kbl(). When used in HTML markdown this function sometimes generates unintended behavior, eg converting (part of) strings to email addresses when they contain an @ sign. This functions can replace possible problematic parts with something else. This can be eg latex. For example: replace '@' with '$@$' will solve the email address 'problem'

for obvious reasons only character vector columns are processed

---

`getAcquistionDate` *function to retrieve the acquisition date of the files used to generate the pdResult file*

Description

function to retrieve the acquisition date of the files used to generate the pdResult file

Usage

```
getAcquistionDate(db)
```

Arguments

`db`	database access 'handle'

Value

one or more POSIXct/POSIXt object(S)

Note

this function is essentially a wrapper around `getAcquistionDateTime`

`getAcquistionDateTime` *function to retrieve the acquisition date & time of the files used to generate the pdResult file*

Description

function to retrieve the acquisition date & time of the files used to generate the pdResult file

Usage

```
getAcquistionDateTime(
  db,
  useAmPm = TRUE,
  format = ifelse(useAmPm, "%m/%d/%Y %I:%M:%S %p", "%m/%d/%Y %H:%M:%S %p")
)
```

Arguments

`db`	database access 'handle'
`useAmPm`	logical, influences what default format is used. Ignored if a format is specified
`format`	character vector specifying the format of the resulting POSIXct/POSIXt object. See `strptime` for more info

Value

one or more POSIXct/POSIXt object(S)

Note

this function is essentially a wrapper around `studyDefinitionFileSets`

`getBlobs` *detemines which columns in a table are of the blob (raw) type*

Description

detemines which columns in a table are of the blob (raw) type

Usage

```
getBlobs(theTable)
```

Arguments

`theTable`	the table containing the data

Value

a data.frame with two columns: name = colum name) and type (which should always be 'blob')

Note

meant for use in debugging problems

`getPeptideInfo` *get peptide information from the peptide table from a pdResult file based on the provided proteinAccession (uniprot) codes. Raw columns are "translated"*

Description

get peptide information from the peptide table from a pdResult file based on the provided proteinAccession (uniprot) codes. Raw columns are "translated"

Usage

```
getPeptideInfo(
  db,
  columns = "AbundancesNormalized",
  addStandardColumns = TRUE,
  proteinAccessions = knockOutProteins()$Accession,
  removeUnusedQuantInfo = TRUE
)
```

Arguments

`db`	database access 'handle'
`columns`	allows the selection of columns to take from the table. The columns: PeptideGroupID, Sequence, Modifications, QuanInfo are automatically included. Default column to be retrieved is AbundancesNormalized
`addStandardColumns`	if TRUE then the following columns are added by default to the columnNames argument: "PeptideGroupID", "Sequence", "Modifications" & "QuanInfo". Please note that this will give problems if these columns are also in the columnNames argument. Also: to be able to use the argument removeUnusedQuantInfo = TRUE, you MUST retrieve the "QuantInfo" column
`proteinAccessions`	defines from which protein(s) info will be retrieved (character vector)
`removeUnusedQuantInfo`	default = TRUE. IF TRUE then only peptide info rows with NA as QuantInfo are kept (the others contain problematic abundance info or none at all)

Value

a named list of data.frames (the names are the proteinAccessions)

Note

this function uses the default `getProteinInfoRaw` function. If more control over the "translation" of raw columns is needed, then use `getPeptideInfoRaw` and do the translation manually

---

`getPeptideInfoRaw` *get peptide information from the peptide table from a pdResult file based on the provided proteinAccession (uniprot) codes. Raw columns are not "translated"*

Description

get peptide information from the peptide table from a pdResult file based on the provided proteinAccession (uniprot) codes. Raw columns are not "translated"

Usage

```
getPeptideInfoRaw(
  db,
  columns = "AbundancesNormalized",
  addStandardColumns = TRUE,
  proteinAccessions = knockOutProteins()$Accession
)
```

Arguments

`db`	database access 'handle'
`columns`	allows the selection of columns to take from the table. The columns: PeptideGroupID, Sequence, Modifications, QuanInfo are automatically included. Default column to be retrieved is AbundancesNormalized
`addStandardColumns`	if TRUE then the following columns are added by default to the columnNames argument: "PeptideGroupID", "Sequence", "Modifications" & "QuanInfo". Please note that this will give problems if these columns are also in the columnNames argument
`proteinAccessions`	defines from which protein(s) info will be retrieved (character vector)

Value

a named list of data.frames (the names are the proteinAccessions) `getProteinInfo` *get protein info (with translation of columns) from a list of protein Accessions (uniprot code). Essentially this is a wrapper function for* `getProteinInfoRaw`

Description

get protein info (with translation of columns) from a list of protein Accessions (uniprot code). Essentially this is a wrapper function for `getProteinInfoRaw`

Usage

```
getProteinInfo(
  db,
 columns = c("Accession", "ProteinGroupIDs", "AbundancesNormalized", "AbundanceRatios",
    "AbundanceRatioPValue", "AbundanceRatioAdjPValue"),
  proteinAccessions = knockOutProteins()$Accession,
  sortorder = "Accession"
)
```

Arguments

`db`	database access 'handle'
`columns`	allows the selection of columns to take from the table
`proteinAccessions`	defines which protein(s) info will be retrieved (character vector)
`sortorder`	allows for sorting of the resulting data.frame by on of it's columns (default = "Accession")

Value

a data.frame containing requested data from the protein table after "translation" of the raw columns

Note

this function uses the default `getProteinInfoRaw` function. If more control over the "translation" of raw columns is needed, then use `getProteinInfoRaw` and do the translation manually

getProteinInfoRaw

get protein info (without translation of columns) from a list of protein Accessions (uniprot code). Essentially this is a wrapper function for `dbGetTable`

Description

get protein info (without translation of columns) from a list of protein Accessions (uniprot code). Essentially this is a wrapper function for `dbGetTable`

Usage

```
getProteinInfoRaw(
  db,
 columns = c("Accession", "ProteinGroupIDs", "AbundancesNormalized", "AbundanceRatios",
    "AbundanceRatioPValue", "AbundanceRatioAdjPValue"),
  proteinAccessions = knockOutProteins()$Accession,
  sortorder = "Accession",
  SQL = FALSE
)
```

Arguments

`db`	database access 'handle'
`columns`	allows the selection of columns to take from the table
`proteinAccessions`	defines which protein(s) info will be retrieved (character vector)
`sortorder`	allows for sorting of the resulting data.frame by on of it's columns (default = "Accession")
`SQL`	allows the function to return the SQL query statement in stead of a data.frame (for debugging purposes)

Value

a data.frame containing requested data from the protein table or a character string specifying an SQL query

isMasterProtein

function for 'translation' of the isMasterProtein values (0..4) in the proteinTable to words (like in Proteome Discoverer).

Description

function for 'translation' of the isMasterProtein values (0..4) in the proteinTable to words (like in Proteome Discoverer).

Usage

```
isMasterProtein(info)
```

Arguments

`info`	integer vector to be ’translated’

Value

character vector (the translation)

`knockOutProteins` *helper function to generate the a data.frame of proteins info for other functions*

Description

helper function to generate the a data.frame of proteins info for other functions

Usage

```
knockOutProteins()
```

Value

a data.frame with three columns: short (character vector), Accession (character vector, uniprot "style") and knockout (logical)

Note

even though it’s called knockOutProteins, 2 of the proteins are not knock out proteins.

`MSfileInfo` *get the table with info on the files used in the search from the database*

Description

get the table with info on the files used in the search from the database

Usage

```
MSfileInfo(db, type = "XcaliburRawfile", dates = thermo.date, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`type`	allows for selection of the FileTypes default = "XCaliburRawFile"
`dates`	allows transformation of the date/time strings from te database to be transformed into proper data/time fields. Default function used is thermo.date. If no transformation is required, use na.date
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

data.frame

`na.date` *fake converter for times when no conversion is wanted/needed*

Description

fake converter for times when no conversion is wanted/needed

Usage

```
na.date(theDate)
```

Arguments

`theDate`	character string (can be vectorized)

Value

theDate (original character string)

`nodes` *function that takes a (xmlToList type) workflow and returns a list of nodes*

Description

function that takes a (xmlToList type) workflow and returns a list of nodes

Usage

```
nodes(
  workflow,
  showHidden = FALSE,
  showAdvanced = TRUE,
  showConfiguration = FALSE
)
```

Arguments

`workflow`	a (xmlToList type) workflow
`showHidden`	if TRUE then rows with hidden = TRUE are included (default: false)
`showAdvanced`	if TRUE then rows with advanced = TRUE are included (default: TRUE)
`showConfiguration`	if TRUE then rows with configuration = TRUE are included (default: FALSE)

Value

a list of named data.frame objects containing all the parameters/ settings in the nodes of the workflow

Note

an example of it's use: (workflowInfo(db))$nodeInfo$Consensus nodes()

`nodeTable` *function to display an overview table of the processing/consensus workflows in the nodeInfo coming out of the workflowInfo function*

Description

function to display an overview table of the processing/consensus workflows in the nodeInfo coming out of the workflowInfo function

Usage

```
nodeTable(nodeInfo)
```

Arguments

`nodeInfo`	either the processing or consensus part of the nodeInfo

Note

an example of it's use: (workflowInfo(db))$nodeInfo$Consensus

---

`pQuanInfo` *function for translation of the QuanInfos values in the psms & peptide tables to words (like in Proteome Discoverer).*

Description

function for translation of the QuanInfos values in the psms & peptide tables to words (like in Proteome Discoverer).

Usage

```
pQuanInfo(info)
```

Arguments

`info`	integer vector to be 'translated'

Value

character vector (the translation)

`proteinIDTypes` *get the names of the identification types (sequest HT etc) used in the database*

Description

get the names of the identification types (sequest HT etc) used in the database

Usage

```
proteinIDTypes(db, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`SQL`	allows the function to return the SQL query statement in stead of a data.frame

Value

data.frame with a single column: "GroupName" `psmAmbiguity` *function for 'translation' of the psmAmbiguity values (1..5) in the psmTable to words (like in Proteome Discoverer). <...> –> means not encountered/ undefined/no inference*

Description

function for 'translation' of the psmAmbiguity values (1..5) in the psmTable to words (like in Proteome Discoverer). <...> –> means not encountered/ undefined/no inference

Usage

```
psmAmbiguity(info)
```

Arguments

`info`	integer vector to be 'translated'

Value

character vector (the translation)

`quanInfo` *function for 'translation' of the QuanInfo values in the QuanSpectrumInfo table to words (like in Proteome Discoverer).*

Description

function for 'translation' of the QuanInfo values in the QuanSpectrumInfo table to words (like in Proteome Discoverer).

Usage

```
quanInfo(info)
```

Arguments

`info`	integer vector to be 'translated'

Value

character vector (the translation)

quanInfoDetails — *function for 'translation' of the QuanInfoDetails values in the QuanSpectrumInfo table to words (like in Proteome Discoverer).*

Description

function for 'translation' of the QuanInfoDetails values in the QuanSpectrumInfo table to words (like in Proteome Discoverer).

Usage

```
quanInfoDetails(info)
```

Arguments

`info`	integer vector to be 'translated'

Value

character vector (the translation)

replacementStrings — *function that generates the default data.frame for the function df_replace().*

Description

function that generates the default data.frame for the function df_replace().

Usage

```
replacementStrings()
```

Value

a data.frame with columns: value, replacement and singleChar

Note

value is the string to be searched, replacement is what it needs to be replaced with. singleChar sets whether the replacement should only take place when dealing with single character strings. This is because single character strings sometimes 'act' different when rendering markdown documents in HTML

SearchInfo *get the table with info on the search itself from the database*

Description

get the table with info on the search itself from the database

Usage

```
SearchInfo(db, SQL = FALSE)
```

Arguments

db	database access 'handle'
SQL	allows the function to return the SQL query statement in stead of a data.frame

Value

data.frame

spectrum.centroid *gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum centroided spectrum*

Description

gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum centroided spectrum

Usage

```
spectrum.centroid(spectrum)
```

Arguments

spectrum	list object containing info on a spectrum

Value

data.frame

`spectrum.header` *gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum header*

Description

gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum header

Usage

```
spectrum.header(spectrum)
```

Arguments

`spectrum`	list object containing info on a spectrum

Value

data.frame

`spectrum.precursor.additionalInfo` *gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent additonal info*

Description

gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent additonal info

Usage

```
spectrum.precursor.additionalInfo(spectrum)
```

Arguments

`spectrum`	list object containing info on a spectrum

Value

data.frame

`spectrum.precursor.centroid` *gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent centroided spectrum*

Description

gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent centroided spectrum

Usage

```
spectrum.precursor.centroid(spectrum)
```

Arguments

`spectrum`	list object containing info on a spectrum

Value

data.frame

`spectrum.precursor.header` *gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent header*

Description

gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent header

Usage

```
spectrum.precursor.header(spectrum)
```

Arguments

`spectrum`	list object containing info on a spectrum

Value

data.frame

spectrum.precursor.info *gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent monoisotopic peak*

Description

gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent monoisotopic peak

Usage

```
spectrum.precursor.info(spectrum, measured = TRUE)
```

Arguments

`spectrum`	list object containing info on a spectrum
`measured`	logical vector, if TRUE then the measured data is returned

Value

data.frame

spectrum.precursor.profile *gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent profile spectrum*

Description

gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent profile spectrum

Usage

```
spectrum.precursor.profile(spectrum)
```

Arguments

`spectrum`	list object containing info on a spectrum

Value

NULL or NA

Note

this is a convenience function, type of data not observed

`spectrum.precursor.scanEvent` *gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent scan event*

Description

gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum parent scan event

Usage

```
spectrum.precursor.scanEvent(spectrum, returnRaw = FALSE)
```

Arguments

`spectrum`	list object containing info on a spectrum
`returnRaw`	logical vector, if TRUE them the data is returned as a list. if FALSE (default) then a data.frame of all character-type data is returned'

Value

data.frame or list

`spectrum.profile` *gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum profile spectrum*

Description

gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum profile spectrum

Usage

```
spectrum.profile(spectrum)
```

Arguments

`spectrum`	list object containing info on a spectrum

Value

NULL or NA

Note

this is a convenience function, type of data not observed `spectrum.scanEvent` *gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum scan event*

Description

gets the info in the list object coming from the function 'transformSpectrumRaw': spectrum scan event

Usage

```
spectrum.scanEvent(spectrum)
```

Arguments

`spectrum`	list object containing info on a spectrum

Value

data.frame

`studyDefinitionExtensions` *function that extracts information on isotope corrections (if available)*

Description

function that extracts information on isotope corrections (if available)

Usage

```
studyDefinitionExtensions(analysisDef, correctXML = c("utf-16", "utf-8"))
```

Arguments

`analysisDef`	generated by the analysisDefinition function
`correctXML`	can only have two different values: NA or a two element character vector c("utf-16","utf-8"). During the research into the method descriptions in the XML object it was noticed that the XML::xmlToList gave an error Document labelled UTF-16 but has UTF-8 content. This was solved by replacing the 'utf-16' string by 'utf-8' string in the XML object. This may be a country specific issue, so the function allows setting this parameter to NA will not do the replacement.

Value

NA or a list of two data.frame objects

`studyDefinitionExtensionSettings` *function to extract sample/factor/ratio/replicate information.*

Description

function to extract sample/factor/ratio/replicate information.

Usage

```
studyDefinitionExtensionSettings(analysisDef)
```

Arguments

`analysisDef`	generated by the analysisDefinition function

Value

a lits of 4 elements:

1. StudyVariablesForGrouping : a data.frame of factors used
2. StudyVariablesForSorting : a data.frame of sorting specification for the factors
3. QuanRatios : a list object of all ratios. Each ratio has the following elements: RatioTable (specifying numerator/denominator), RatioString (for easy info printing), NumeratorSamples & DenominatorSamples specifying which samples are in the numerator and denominator and finally Replicates which contains info on replicates.
4. XML : the actual from which the information comes. This was included because the exact specification for all possible cases is not (yet) known

Note

So far, this function has not been tested for all possible cases/ scenarios.

`studyDefinitionFactors` *function that extracts the factors used in the study to generate the .pdResult file. The result contains some internal info in the form of columns named id (identifiers).*

Description

function that extracts the factors used in the study to generate the .pdResult file. The result contains some internal info in the form of columns named id (identifiers).

Usage

```
studyDefinitionFactors(analysisDef)
```

Arguments

`analysisDef`	generated by the analysisDefinition function

Value

data.frame with the info

`studyDefinitionFileSets`
function that extracts file information on the original .raw files used to generate the .pdResult file. Information includes the original file name, location & size. It also contains some internal info in the form of columns named id (identifiers).

Description

function that extracts file information on the original .raw files used to generate the .pdResult file. Information includes the original file name, location & size. It also contains some internal info in the form of columns named id (identifiers).

Usage

```
studyDefinitionFileSets(analysisDef, splitFileSize = TRUE, joinedTables = TRUE)
```

Arguments

`analysisDef`	generated by the analysisDefinition function
`splitFileSize`	boolean (default: TRUE), specifies if the FileSize column should be split into the actual file size (still a character vector) and the file size format
`joinedTables`	boolean (default: TRUE), specifies if all info should be put in a single data.frame. If FALSE it will generate a list of two data.frame objects; this might be useful in some scenarios

Value

data.frame or list of two data.frame objects

`studyDefinitionQuanMethods`
function that extracts quantification method information if a quantification method was used to generate the .pdResult file

Description

function that extracts quantification method information if a quantification method was used to generate the .pdResult file

Usage

```
studyDefinitionQuanMethods(analysisDef)
```

Arguments

`analysisDef`	generated by the analysisDefinition function

Value

A list of two data.frame objects. The first one will contain the name, description, etc. The second one will specify the names of the labels used. The result will be NA in the case that no quantification method was used.

`studyDefinitionSamples`
function that extracts sample information. The information seems to be a bit redundant, as the info is also seen in other tables.

Description

function that extracts sample information. The information seems to be a bit redundant, as the info is also seen in other tables.

Usage

```
studyDefinitionSamples(analysisDef)
```

Arguments

`analysisDef`	generated by the analysisDefinition function

Value

a data.frame

system.date *converts character string date into date/time format*

Description

converts character string date into date/time format

Usage

```
system.date(theDate, dateFormat = lubridate::ymd_hms)
```

Arguments

`theDate`	character string to be converted (can be vectorized)
`dateFormat`	function that defines the output date/time format, default is lubridate::ymd_hms

Value

date

tableNames *internal helper function to prevent having to remember the somewhat long names of the most used tables*

Description

internal helper function to prevent having to remember the somewhat long names of the most used tables

Usage

```
tableNames(whichTable = "proteins")
```

Arguments

`whichTable`	can be either "proteins","peptides","psms" or "consensus" character do not need to be lower or upper case (all are converted to upper case). If another string is used as a parameter, the function will return NA

Value

a string containing the protein discoverer table name corresponding to the parameter whichTable

thermo.date *converts character string date into date/time format*

Description

converts character string date into date/time format

Usage

```
thermo.date(theDate, dateFormat = lubridate::mdy_hms)
```

Arguments

`theDate`	character string to be converted (can be vectorized)
`dateFormat`	function that defines the output date/time format, default is lubridate::mdy_hms

Value

date in mdy hms format

tmt10Channels *helper function to generate the a data.frame of TMT knockout strain (TKO) info for other functions. This function generates a data.frame based on the 10-plex TMT TKO knockout (this was the original TMT-knockout-digest available)*

Description

helper function to generate the a data.frame of TMT knockout strain (TKO) info for other functions. This function generates a data.frame based on the 10-plex TMT TKO knockout (this was the original TMT-knockout-digest available)

Usage

```
tmt10Channels()
```

Value

a data.frame with four columns: all are character vectors

Note

the rows define the order of the abundance (etc) columns in the protein, peptide and psms table in a pdResult file. The order is alphabetical in protein & peptide tables, but not in the psms tables: there it is based based on the order of the isotopes

psmsChannels & isotopeChannels columns match each other

`tmt11Channels`	*helper function to generate the a data.frame of TMT knockout strain (TKO) info for other functions. This function generates a data.frame based on the 11-plex TMT TKO knockout*

Description

helper function to generate the a data.frame of TMT knockout strain (TKO) info for other functions. This function generates a data.frame based on the 11-plex TMT TKO knockout

Usage

```
tmt11Channels()
```

Value

a data.frame with four columns: all are character vectors

Note

the rows define the order of the abundance (etc) columns in the protein, peptide and psms table in a pdResult file. The order is alphabetical in protein & peptide tables, but not in the psms tables: there it is based based on the order of the isotopes

psmsChannels & isotopeChannels columns match each other

`totalSearchTime`	*get the total search time from the database*

Description

get the total search time from the database

Usage

```
totalSearchTime(db, SQL = FALSE)
```

Arguments

`db`	database access 'handle'
`SQL`	allows the function to return the SQL query statement used

Value

numeric: search time in seconds

transformSpectrumRaw *transforms a spectrum from the table 'MassSpectrumItems' into a R compatible list*

Description

transforms a spectrum from the table 'MassSpectrumItems' into a R compatible list

Usage

```
transformSpectrumRaw(spectrumObject)
```

Arguments

`spectrumObject`	must be of class 'raw'

Value

a list object containing info on the spectrum (object). This list object can be further translated via the function 'translateSpectrumInfo'

Note

this functions writes a temporary file tp disk, which is unzipped, read and deleted again

workflowInfo *function to get the workflow information from a .pdResult file*

Description

function to get the workflow information from a .pdResult file

Usage

```
workflowInfo(db, workflowsTable = "WorkFlows", returnNodeData = TRUE)
```

Arguments

`db`	database access 'handle' pointing to a .pdResult file
`workflowsTable`	name of the table containing the info. Default is 'WorkFlows'
`returnNodeData`	if TRUE then the node parameters are included in the returned data

Value

either a single data.frame containing basic info on the workflows or (if returnNodeData is TRUE) a list of the data.frame with the second list element containing information on the nodes that make up the processing & the consensus workflows (in xmlToList result format). This second element (called nodeInfo) is used in additional functions to show/display the processing/consensus workflows.

Index