padmet Documentation

Meziane AITE

Oct 26, 2020

Contents

CHAPTER 1

Installation

Stable releases of padmet are best installed via `pip` or `easy_install` from PyPI using something like:

```
$ pip install padmet
```

CHAPTER 2

Usage

To get started using padmet, install the library as described above. Once the library becomes available on the given system, it can be developed against. The developed scripts do not need to reside in any particular location on the system.

CHAPTER 3

Development

Anyone interested in contributing or tweaking the library is more then welcome to do so. To start, simply fork the Git repository on Github and start playing with it. Then, issue pull requests.

CHAPTER 4

Tutorial

4.1 Tutorial

4.1.1 PADMet format

Padmet is an object representing the metabolic network of a species (organism) based on a reference database.

This format contains Node, Policy and Relation:

- a Node is an Object that contains information about an element of the network (can be a pathway, reaction...).
- a Policy defines the way Node and Relation are associated.
- a Relation defines how two nodes are connected. In a relation there is a node "in" and a node "out". (reactionX (**node in**) consumes metaboliteX (**node out**))

A Padmet object contains 3 attributes:

- dicOfNode: a dictionary of node: key=Node's unique id / value = <Node>
- dicOfRelationIn: a dictionary of relation with: key= nodeIN id / value = list of <relation>
- dicOfRelationOut: a dictionary of relation with: key= nodeOut id / value = list of <relation>

4.1.2 Node

A Node represent an element in a metabolic network.

A Node has a type ('reaction','pathway'), an ID and a dictionary of miscellaneous data. Node can be find in the dicOfNode, this dicitonary contains Node IDs as key and Node object as values.

As an example we use the padmet_1.padmet from test data:

```
>>> from padmet.classes import PadmetSpec

# Creation of a padmet object from the padmet file.
```

(continues on next page)

(continued from previous page)

```
>>> padmet_object = PadmetSpec('test_data/padmet/padmet_1.padmet')

# Selection of the Node corresponding to the reaction 'ENOYL-COA-HYDRAT-RXN'
>>> enoyl_coa_node = padmet_object.dicOfNode['ENOYL-COA-HYDRAT-RXN']

>>> enoyl_coa_node.id
'ENOYL-COA-HYDRAT-RXN'

>>> enoyl_coa_node.type
'reaction'

>>> enoyl_coa_node.misc
{'EC-NUMBER': ['EC-4.2.1.17'], 'DIRECTION': ['LEFT-TO-RIGHT']}
```

4.1.3 Relation

A Relation represent a link between two elements (node) in a metabolic network.

A Relation contains 4 attributes:

- type: The type of the relation (e.g: 'consumes' or 'produces')
- id_in: the identifier of the node corresponding to the subject of the relation (e.g: 'RXN-1')
- id_out: the identifier of the node corresponding to the object of the relation (e.g: 'CPD-1')
- misc: A dictionary of miscellaneous data, k = tag of the data, v = list of values (e.g: {'STOICHIOMETRY':[1.0]})

We will use the same example as the Node section:

```
>>> from padmet.classes import PadmetSpec

# Creation of a padmet object from the padmet file.
>>> padmet_object = PadmetSpec('test_data/padmet/padmet_1.padmet')

# Selection of the Node corresponding to the reaction 'ENOYL-COA-HYDRAT-RXN'
>>> enoyl_coa_relation = padmet_object.dicOfRelationIn['ENOYL-COA-HYDRAT-RXN']

>>> len(enoyl_coa_relation)
6
# Node 'ENOYL-COA-HYDRAT-RXN' is an ID in for 6 Relations

# What is the type of the first relation when this Node is In
>>> padmet_object.dicOfRelationIn['ENOYL-COA-HYDRAT-RXN'][0].type
'is_in_pathway'

>>> padmet_object.dicOfRelationIn['ENOYL-COA-HYDRAT-RXN'][0].id_in
'ENOYL-COA-HYDRAT-RXN'

>>> padmet_object.dicOfRelationIn['ENOYL-COA-HYDRAT-RXN'][0].id_out
'FAO-PWY'

>>> padmet_object.dicOfRelationIn['ENOYL-COA-HYDRAT-RXN'][0].toString()
'ENOYL-COA-HYDRAT-RXN\tis_in_pathway\tFAO-PWY'
```

4.1.4 Create padmet

Padmet files can be created using the padmet `padmet pgdb_to_padmet` command. This command takes as input the result of Pathway Tools software. You can get these files by using the mpwt package.

```
usage:
    padmet pgdb_to_padmet --pgdb=DIR --output=FILE [--version=V] [--db=ID] [-
↪-padmetRef=FILE] [--source=STR] [-v] [--enhance]
    padmet pgdb_to_padmet --pgdb=DIR --output=FILE --extract-gene [--no-
↪orphan] [--keep-self-rxn] [--version=V] [--db=ID] [--padmetRef=FILE] [--
↪source=STR] [-v] [--enhance]

options:
    -h --help     Show help.
    --version=V    Xcyc version [default: N.A].
    --db=ID    Biocyc database corresponding to the pgdb (metacyc, ecocyc, ..
↪.) [default: N.A].
    --output=FILE    padmet file corresponding to the DB.
    --pgdb=DIR    directory containg all the .dat files of metacyc (data).
    --padmetRef=FILE    padmet of reference.
    --source=STR    Tag associated to the source of the reactions, used to␣
↪ensure traceability [default: GENOME].
    --enhance    use the metabolic-reactions.xml file to enhance the␣
↪database.
    --extract-gene    extract genes from genes_file (use if its a specie's␣
↪pgdb, if metacyc, do not use).
    --no-orhpan    remove reactions without gene associaiton (use if its a␣
↪specie's pgdb, if metacyc, do not use).
    --keep-self-rxn    remove reactions with no reactants (use if its a␣
↪specie's pgdb, if metacyc, do not use).
    -v   print info.
```

The input folder (–pgdb) should be like:

```
input_folder
├── classes.dat
├── compound-links.dat
├── compounds.dat
├── dnabindsites.dat
├── enzrxns.dat
├── gene-links.dat
├── genes.dat
├── pathway-links.dat
├── pathways.dat
├── promoters.dat
├── protein-features.dat
├── protein-links.dat
├── proteins.dat
├── protligandcplxes.dat
├── pubs.dat
├── reaction-links.dat
├── reactions.dat
├── regulation.dat
├── regulons.dat
├── rnas.dat
├── species.dat
├── terminators.dat
└── transunits.dat
```

This command will create a padmet file.

CHAPTER 5

API Documentation

5.1 Classes API

5.1.1 API documentation for padmet.classes

Node

class padmet.classes.node.**Node**(*_type*, *_id*, *misc=None*)

A Node represent an element in a metabolic network

e.g: compound, reaction.

Parameters

- **_type** (*str*) – The type of the node ('reaction','pathway')
- **_id** (*str*) – the identifier of the node ('rxn-45)
- **misc** (*dict*) – A dictionary of miscellaneous data ({'DIRECTION':[REVERSIBLE]}) (the default value is None)

toString()

This function is used to stock the information relative to the node in a padmet file.

Returns string with all data sep by tab' ex: reaction RXN0..

Return type str

Policy

class padmet.classes.policy.**Policy**(*policy_in_array=None*)

A Policy define the types of relations and nodes of a network.

A policy contains 3 attributes:

policy_in_array: Is a list of list of relations

e.g: [['reaction','consumes','compounds'],['reaction','produces','compounds']]

class_of_node: Is a set of all the type of nodes represented in the network

e.g: set(['reaction', 'compound'])

type_of_arc: Is a dictionary of all the types of arcs represented in the network

(e.g: {'reaction':['consumes','compounds']})

Parameters **policy_in_array** (*list*) – Is a list of list of relations

e.g: [['reaction','consumes','compounds'],['reaction','produces','compounds']]

(the default value is None)

getClassOfNode()

Returns return class_of_node

Return type set

getPolicyInArray()

Returns return policy_in_array

Return type list

getTypeOfArc()

Returns return type_of_arc

Return type dict

setPolicyInArray(*policy_in_array*)

From policy_in_array, set class_of_node and type_of_arc

Parameters **policy_in_array** (*list*) – Is a list of list of arcs. e.g: [['reaction','consumes','compounds'],['reaction','produces','compounds']]

Relation

class padmet.classes.relation.**Relation**(*id_in*, *_type*, *id_out*, *misc=None*)

A Relation represent a link between two elements (node) in a metabolic network.

e.g: RXN-1 consumes CPD-1

A Relation contains 4 attributes:

_type: The type of the relation (e.g: 'consumes' or 'produces')

id_in: the identifier of the node corresponding to the subject of the relation (e.g: 'RXN-1)

id_out: the identifier of the node corresponding to the object of the relation (e.g: 'CPD-1)

misc: A dictionary of miscellaneous data, k = tag of the data, v = list of values

(e.g: {'STOICHIOMETRY':[1.0]})

Parameters

- **id_in** (*str*) – the identifier of the node corresponding to the subject of the relation ('RXN-1)

- **_type** (*str*) – The type of the relation (e.g: 'consumes' or 'produces')
- **id_out** (*str*) – the identifier of the node corresponding to the object of the relation ('CPD-1)
- **_misc** (*dict*) – A dictionary of miscellaneous data (e.g: {'STOICHIOMETRY':[1.0]})

compare(*relation*)
compare 2 relations. First check if ids and type are the same, then check the misc dictionary. :param relation: the relation to compare :type relation: padmet.Relation

Returns Return True if relation are the same, False if not

Return type bool

toString()
This function is used to stock the information relative to the node in a padmet file.

Returns string with all data sep by tab' ex: RXN0 consumes CPD-A..

Return type str

PadmetRef

class padmet.classes.padmetRef.**PadmetRef**(*padmetRef_file=None*)
PadmetRef is an object representing a DATABASE of metabolic network.

Contains <Policy>, <Node> and <Relation>:

The policy defines the way Node and Relation are associated.

A node is an Object that contains information about an element of the network (can be a pathway, reaction...).

A realtion defines how two nodes are connected. In a relation there is a node "in" and a node "out". (reactionX'in' consumes metaboliteX'out').

PadmetRef contains 3 attributs: dicOfNode: a dictionary of node: key=Node's unique id / value = <Node>.

dicOfRelationIn: a dictionnary of relation with: key= nodeIN id / value = list of <relation>.

dicOfRelationOut: a dictionnary of relation with: key= nodeOut id / value = list of <relation>.

policy: a <policy>.

info: a dictionnary of informations about the network, the database used... This dictionnary is always represented in the header of a padmet file.

if None, initializes an empty <PadmetRef>

Parameters **padmetRef_file** (*str*) – pathname of the padmet file

createNode(*_type*, *_id*, *dicOfMisc={}*, *listOfRelation=None*)
Creation of new node to add in the network.

Parameters

- **_type** (*str*) – type of node (gene, reaction...)
- **_id** (*str*) – id of the node
- **dicOfMisc** (*dict*) – dictionnary of miscellaneous data
- **listOfRelation** (*list or None*) – list of relation

Returns new_node

Return type padmet.classes.Node

delNode(*node_id*)
Allows to delete a node, the relations associated to the node, and for some relations, delete the associated node. For relations where the node to del is 'in':

if rlt type in ['has_xref','has_name','has_suppData']: delNode out

For relations where the node to del is 'out': if rlt type in ['consumes','produces']

Parameters **node_id** (*str*) – id of node to delete

Returns True if node successfully deleted, False if node not in dicOfNode

Return type bool

generateFile(*output*)
Allow to create a padmet file to stock all the data. @param output: pathname of the padmet file to create

Parameters **output** (*str*) – path to output file

getAllRelation()

Returns return a set of all relations

Return type set

loadGraph(*padmet_file*)
Allow to recover all the informations of the padmet file. The section Data Base informations corresponds to the self.info The section Nodes corresponds to the data of each nodes in self.dicOfNode, sep =" " The section Relations corresponds to the data of each relations in self.dicOfRelationIn/Out, sep =" "

Parameters **padmet_file** (*str*) – the pathname of the padmet file to load.

setDicOfNode(*source*)
Set dicOfNode from a dict or copying from an other padmet

Parameters **source** (*dict or padmet.classes.PadmetRef*) – may be a dict or an other padmet from where will be copied the dicOfNode

setInfo(*source*)
All the information printed in the header of a padmet stocked in a dict. {"metacyc":{version:XX,...},"ecocyc":{...}...} set Info from a dictionnary or copying from an other padmet

Parameters **source** (*dict or padmet.classes.PadmetRef*) – may be a dict or an other padmet from where will be copied the info

setPolicy(*source*)
Set policy from a list or copying from an other padmet

Parameters **source** (*list or padmet.classes.PadmetRef*) – may be a list or an other padmet from where will be copied the policy

setdicOfRelationIn(*source*)
Set dicOfRelationIn from a dict or copying from an other padmet

Parameters **source** (*dict or padmet.classes.PadmetRef*) – may be a dict or an other padmet from where will be copied the dicOfRelationIn

setdicOfRelationOut(*source*)
Set dicOfRelationOut from a dict or copying from an other padmet

Parameters **source** (*dict or padmet.classes.PadmetRef*) – may be a dict or an other padmet from where will be copied the dicOfRelationOut

updateFromSbml (*sbml_file*, *verbose=False*)
Initialize a padmetRef from sbml. Copy all species, convert id with sbmlPlugin stock name in COMMON NAME. Copy all reactions, convert id with sbmlPlugin, stock name in common name, stock compart and stoichio data relative to reactants and products in the misc of consumes/produces relations

Parameters

- **sbml_file** (*str*) – pathname of the sbml file
- **verbose** (*bool*) – if True print supp info

PadmetSpec

class padmet.classes.padmetSpec.**PadmetSpec** (*padmetSpec_file=None*)
PadmetSpec is an object representing the metabolic network of a species(organism) based on a reference database PadmetRef. contains <Policy>, <Node> and <Relation> The policy defines the way Node and Relation are associated A node is an Object that contains information about an element of the network (can be a pathway, reaction...). A relation defines how two nodes are connected. In a relation there is a node "in" and a node "out". (reactionX'in' consumes metaboliteX'out') PadmetSpec contains 3 attributes:

dicOfNode: a dictionary of node: key=Node's unique id / value = <Node> dicOfRelationIn: a dictionary of relation with: key= nodeIN id / value = list of <relation> dicOfRelationOut: a dictionary of relation with: key= nodeOut id / value = list of <relation> policy: a <policy> info: a dictionary of informations about the network, the database used... This dictionary is always represented in the header of a padmet file

if None, initializes an empty <PadmetSpec>

Parameters **padmetSpec_file** (*str*) – pathname of the padmet file

change_compart (*old_compart*, *new_compart*, *verbose=False*)
#TODO

copyNode (*padmet*, *node_id*)
copyNode() allows to copy a node from an other padmetSpec or padmetRef. It copies all the relations 'in' and 'out' and it calls the function _copyNodeExtend() to recover the associated node.

Parameters

- **Padmet** (*padmet.classes.PadmetRef/PadmetSpec*) – padmet from where to copy the node
- **node_id** (*str*) – the id of the node to copy

createNode (*_type*, *_id*, *dicOfMisc={}*, *listOfRelation=None*)
Creation of new node to add in the network.

Parameters

- **_type** (*str*) – type of node (gene, reaction...)
- **_id** (*str*) – id of the node
- **dicOfMisc** (*dict*) – dictionnary of miscellaneous data
- **listOfRelation** (*list or None*) – list of relation

Returns new_node

Return type padmet.classes.Node

delCompart(*compart*, *verbose=False*)
#TODO

delNode(*node_id*)
Allows to delete a node, the relations associated to the node, and for some relations, delete the associated node. For relations where the node to del is 'in':

if rlt type in ['has_xref','has_name','has_suppData']: delNode out

For relations where the node to del is 'out': if rlt type in ['consumes','produces']

Parameters **node_id** (*str*) – id of node to delete

Returns True if node successfully deleted, False if node not in dicOfNode

Return type bool

extract_pathway(*node_id*, *padmetRef_file*, *output*, *sbml=None*)
Allow to extract a pathway in a csv file. Need a padmetRef to check the total number of reactions in the pathway. Header = "Reactions (metacyc_id)", "Reactions (common_name)", "EC-Number",

"Formula (metacyc_id)", "Formula (common_name)", "Found in the network"

Parameters

- **node_id** (*str*) – id of the pathway
- **padmetRef_file** (*str*) – pathname of the padmet ref file
- **output** (*str*) – pathname of the output to create
- **sbml** (*None or str*) – if path given, create a sbml file of this pathway

generateFile(*output*)
Allow to create a padmet file to stock all the data.

Parameters **output** (*str*) – path to output file

getAllRelation()
return a set of all relations

Returns return a set of all relations

Return type set

getRelation(*rlt_id_in*, *rlt_type*, *rlt_id_out*)
return a relation

Returns return a relation

Return type *Relation*

getRelationTypeIDIn(*rlt_type*, *rlt_id_in*)
return a relation

Returns return a relation

Return type *Relation*

get_all_compart()
#TODO

get_growth_medium(*b_compart='C-BOUNDARY'*)
return set of metabolites corresponding to the growth medium

ko (*genes*, *verbose=False*)
remove all reactions associated to a given gene or list of genes

Parameters

- **genes** (*str or list*) – one gene or list of genes to remove
- **verbose** (*bool*) – print more info

loadGraph (*padmet_file*)
Allow to recover all the information of the padmet file. The section Data Base information corresponds to the self.info The section Nodes corresponds to the data of each nodes in self.dicOfNode, sep =" " The section Relations corresponds to the data of each relations in self.dicOfRelationIn/Out, sep =" "

Parameters **padmet_file** (*str*) – the pathname of the padmet file to load.

network_report (*output_dir*, *padmetRef_file=None*, *verbose=False*)
Summurizes the network in a folder (output_dir) of 4 files. all_pathways.tsv: report on the pathways of the network. PadmetRef is used to recover the total reactions of a pathways. (sep = " ") line = dbRef_id, Common name, Number of reactions found,

Total number of reaction, Ratio (Reaction found / Total)

all_reactions.tsv: report on the reactions of the network. (sep = " ") line = dbRef_id, Common name, formula (with id),

formula (with common name), in pathways, associated genes, sources

all_metabolites.tsv: report on the metabolites of the network. (sep = " ") line = dbRef_id, Common name, Produced (p), Consumed (c), Both (cp) all_genes.tsv: report on the genes of the network. (sep= " ") line = "id", "Common name", "linked reactions"

Parameters

- **padmetRef_file** (*str*) – pathname of the padmet of reference
- **output_dir** (*str*) – pathname of the folder where to create the reports
- **verbose** (*bool*) – if true print info.

remove_growth_medium (*b_compart='C-BOUNDARY'*, *verbose=False*)
#TODO

setDicOfNode (*source*)
Set dicOfNode from a dict or copying from an other padmet

Parameters **source** (*dict or padmet.classes.PadmetRef/PadmetSpec*) – may be a dict or an other padmet from where will be copied the dicOfNode

setInfo (*source*)
All the information printed in the header of a padmet stocked in a dict. {"metacyc":{version:XX,...},"ecocyc":{...}...} set Info from a dictionary or copying from an other padmet

Parameters **source** (*dict or padmet.classes.PadmetRef/PadmetSpec*) – may be a dict or an other padmet from where will be copied the info

setPolicy (*source*)
Set policy from a list or copying from an other padmet

Parameters **source** (*list or padmet.classes.PadmetRef/PadmetSpec*) – may be a list or an other padmet from where will be copied the policy

set_growth_medium(*new_growth_medium=None*, *padmetRef=None*, *rxn_prefix={'ExchangeSeed', 'TransportSeed'}*, *b_compart='C-BOUNDARY'*, *e_compart='e'*, *c_compart='c'*, *verbose=False*)

if new_growth_medium is None: just remove the growth medium by deleting reactions starting with rxn_prefix else: remove and change by the new growth_medium, a list of compounds.

Parameters

- **new_growth_medium** (*list or None*) – list of metabolites ids for the new media
- **padmetRef** (*str*) – path name of the padmet ref file
- **rxn_prefix** (*set*) – set of prefix corresponding to reactions of exchanges (specific to growth medium)
- **b_compart** (*str*) – ID of the boundary compartment, compound in this compart will have BoundaryCondition True in sbml
- **verbose** (*bool*) – print more info

setdicOfRelationIn(*source*)

Set dicOfRelationIn from a dict or copying from an other padmet

Parameters **source** (*dict or padmet.classes.PadmetRef/PadmetSpec*) – may be a dict or an other padmet from where will be copied the dicOfRelationIn

setdicOfRelationOut(*source*)

Set dicOfRelationOut from a dict or copying from an other padmet

Parameters **source** (*dict or padmet.classes.PadmetRef/PadmetSpec*) – may be a dict or an other padmet from where will be copied the dicOfRelationOut

updateFromPadmet(*padmet*)

update padmet from another padmet file

Parameters **padmet** (*padmet.classes.PadmetRef/PadmetSpec*) – padmet instance

updateFromSbml(*sbml_file*, *padmetRef=None*, *mapping_file=None*, *verbose=False*, *force=False*, *source_tool=None*, *source_category=None*, *source_id=None*)

Copy data from sbml to padmet. If the id of the sbml are different with the padmetRef, need to add a dictionary of associations between the id in the sbml and the id of the database of reference. if no padmetRef, will create reactions and compounds based on the given sbml @param sbml_file: pathname of the sbml file @param padmetRef: <PadmetRef> padmet of reference. @param mapping_dict (opt): pathname of the file containing the association original_id ref_id, sep =" " @param verbose: print info @param force: if true, allow to copy reactions that are not in padmetRef by creating new reaction @type sbml_file, assocIdOriginRef: str @type padmetRef: PadmetRef @type verbose, force: bool @return: _ @rtype: None

updateNode(*node_id*, *data*, *action*)

Allows to update miscellaneous data of a Node. @param node_id: the id of node to update @param data: tuple with data[0] refers to the miscellaneous data key (ex: common_name, direction ...), data[1] is a list of value to add / update. data[1] can be None if the action is to pop the key @param action: if == "add": the list data[1] will be added (ex: adding a common_name) if == "remove": if data[1] is not None, the list data[1] will be removed (ex: remove just one specific common_name) if == "update":data[1] is the new value of the key data[0] ex: updateNode('RXN-5',('direction',['LEFT-TO-RIGHT']),update). The reaction' direction will be change to left2right

Parameters

- **node_id** (*str*) – the id of the node to update
- **data** (*tuple*) – tuple of data to update, data[0] is the key, data[1] is a value, list or None

- **action** (*str*) – action in ['add','pop','remove','update']. Check description for more information

Returns True if node successfully updated

Return type bool

5.2 Utils API

5.2.1 API documentation for padmet.utils

sbmlPlugin

padmet.utils.sbmlPlugin.**ascii_replace**(*match*)
recover banned char from the integer ordinal in the reg.match

padmet.utils.sbmlPlugin.**convert_from_coded_id**(*coded*, *pattern='__'*, *compart_in_id=False*, *reaction_tag='R'*, *species_tag='M'*)
convert an id from sbml format to the original id. try to extract the type of the id and the compart using strong regular expression ex: M_METABOLITE__45__12_c => ('METABOLITE-12', 'M', 'c')

Parameters

- **coded** (*str*) – the encoded id
- **pattern** (*str*) – pattern used to delimit interger ordinal
- **compart_in_id** (*bool*) – if true: the last _* is not mean to be the compart is part of the id
- **reaciton_tag** (*str*) – First letter used to tag a reaction
- **species_tag** (*str*) – First letter used to tag a species

Returns

- *str* – the uncoded id
- *str, None* – type of ID (ex: 'M' or 'R')
- *str, None* – compart of the id

padmet.utils.sbmlPlugin.**convert_to_coded_id**(*uncoded*, *_type=None*, *compart=None*)
convert an id to sbml valid format. First add type of id "R" for reaction "M" for compound at the start and the compart at the end. _type+"_"+uncoded+"_"+compart then replace not allowed char by integer ordinal :param uncoded: the original id to code :type uncoded: str :param _type: the type of the id (ex: 'R' or 'M') :type _type: str :param compart: the compartment of the id (ex: 'c' or 'e') :type compart: str

Returns the coded id

Return type str

padmet.utils.sbmlPlugin.**extractFormula**(*elementR*)
From an SBML reaction_element will return the formula in a string ex: '1.0 FRUCTOSELYSINE_p => 1.0 FRUCTOSELYSINE_c' :param elementR: a reaction from libsbml.element :type elementR: libsbml.element

Returns the formula

Return type str

padmet.utils.sbmlPlugin.**get_all_decoded_version**(*element_id*, *_type*)

Use convert_from_coded function to convert a element_id (reaction or species) _type use define if element is a 'reaction' or un 'species'. Try different decoding combination based on old and new sbml id encoding.

Parameters

- **element_id** (*str*) – the encoded id
- **_type** (*str*) – _type is 'reaction' or 'species'

Returns list of encoded id

Return type list

padmet.utils.sbmlPlugin.**parseGeneAssoc**(*GeneAssocStr*)

Given a grammar of 'and', 'or' and '(' ')'. Extracts genes ids to a list. (geneX and geneY) or geneW' => [geneX,geneY,geneW] :param GeneAssocStr: the string containing genes ids :type GeneAssocStr: str

Returns the list of unique ids

Return type list

padmet.utils.sbmlPlugin.**parseNotes**(*element*)

From an SBML element (ex: species or reaction) will return all the section note in a dictionary. ex: <notes>

<html:body> <html:p>BIOCYC: |Alkylphosphonates|</html:p> <html:p>CHEBI: 60983</html:p>

</html:body>

</notes>

output: {'BIOCYC': **|Alkylphosphonates|**,'CHEBI':'60983'} value is a list in case diff lines for the same type of info

Parameters **element** (*libsbml.element*) – an element from libsbml

Returns the dictionary of note

Return type dict

Grammar-boolean-rapsody

Aim of this script is to produce, for any pattern like:

a&(b|c)&(d|e)

the resulting lists of elements:

(a, b, d) (a, c, d) (a, b, e) (a, c, e)

Use a finite state machine (see generate_fsm() function) to parse the input, produce a syntactic tree from the lexical table, that is finally used for find the expected ouput.

The API consist only in the compile_output(1) function. All other functions are used internally, or not used at all

but conserved for science. (only postfix(1) function is in this last case)

The compile_output(1) function is high-level and simple. You may want to begin there if you want to read these code.

PEP8 is completely busted at some point, but its mainly because of readability of big lines.

Principles: - lexical analysis of the input string, given the lexems (id, operators, parenthesis) - generate the polish notation of the lexical table - syntactic tree construction - walks in the syntactic tree to determine the possible paths to leafs

Idea is, mainly, that a syntree is simple to store (dict {node:successors}), and represent well the input data. The walks is performed by the eval_tree function, and simply consist of a recursive DFS with generation of the whole path since the root each time a leaf is hit. Behavior of AND and OR operators are hardcoded.

Note that there is no real error handling for parenthesis.

```
usage:
    padmet gbr --string=FILE

options:
    -h --help     Show help.
    --string=STR    A string to parse
```

class padmet.utils.gbr.**Command**

Finish = -2

Stack = -1

class padmet.utils.gbr.**Error**

UnexpectedChar = 'unexpected character'

UnexpectedClosing = 'unexpected closing parenthesis'

UnexpectedLetter = 'unexpected letter'

UnexpectedOp = 'unexpected operator'

UnexpectedOpening = 'unexpected opening parenthesis'

class padmet.utils.gbr.**Type**

Closing = '\\)'

EndOfFile = '\x00'

Letter = '[a-zA-Z0-9_\\.:-]'

Op = '[&|]'

Opening = '\\('

Other = ''

padmet.utils.gbr.**command_help**()
Show help for analysis command.

padmet.utils.gbr.**compile_input**(*string*, *combine_or: bool = False*)
Yields the possible combinations parsed from given string. See module docstring for more explanations.

combine_or – if True, will also yield paths (a, b) from 'a|b'

padmet.utils.gbr.**eval_tree**(*tree: dict*, *root: int = 1*, *combine_or: bool = False*) → iter
Yield paths from input syntree, from given root.

combine_or – if True, will also yield paths (a, b) from 'a|b'

padmet.utils.gbr.**gbr_cli**(*command_args*)

padmet.utils.gbr.**generate_fsm**()
Return a dict {previous type: {current type: command/error}}

padmet.utils.gbr.**generate_syntree**(*pretable*)
Return the syntactic tree of given prefix lexical table

padmet.utils.gbr.**lexical_analysis**(*string*)
Generate the lexical table of given string

padmet.utils.gbr.**postfix**(*lextable*)
Return the postfix representation of the given lexical table

Wikipedia definition: While there are tokens to be read:

Read a token. If the token is a number, then add it to the output queue.

If the token is an operator, o1, then:

while there is an operator token, o2, at the top of the operator stack, and either

o1 is left-associative and its precedence is less than or equal to that of o2, or o1 is right associative, and has precedence less than that of o2,

then pop o2 off the operator stack, onto the output queue;

push o1 onto the operator stack.

If the token is a left parenthesis (i.e. "("), then push it onto the stack. If the token is a right parenthesis (i.e. ")"):

Until the token at the top of the stack is a left parenthesis, pop operators off the stack onto the output queue. Pop the left parenthesis from the stack, but not onto the output queue. If the token at the top of the stack is a function token, pop it onto the output queue. If the stack runs out without finding a left parenthesis, then there are mismatched parentheses.

When there are no more tokens to read:

While there are still operator tokens in the stack: If the operator token on the top of the stack is a parenthesis, then there are mismatched parentheses. Pop the operator onto the output queue.

Exit.

padmet.utils.gbr.**precedence**(*op1*, *op2*)
True if op1 have precedence on op2. Here AND have an higher precedence than OR.

padmet.utils.gbr.**prefix**(*lextable*)

- **Scan input in reversed order:**
  - If token is an operand yield it
  - If token is a) push it to stack
  - **If token is an operator o1:**
    - pop from stack and yield each operator o2 that have same or higher precedence than o1.
    - puth o2 to stack
  - **If token is a (:**
    - pop from stack and yield each operator until a) is encountered.
    - Remove the (

padmet.utils.gbr.**type_of**(*character*)
Return the Type value for given character

padmet.utils.gbr.**well_parenthesed**(*lextable*)
True iff given lexical table is well parenthesed

5.2.2 Connection

Description:

#TODO

biggAPI_to_padmet

Description: Require internet access !

Allows to extract the bigg database from the API to create a padmet.

1./ Get all reactions universal id from http://bigg.ucsd.edu/api/v2/universal/reactions, escape reactions of biomass.

2./ Using async_list, extract all the informations for each reactions (compounds, stochio, name . . .)

3./ Need to use sleep time to avoid to lose the server access.

4./ Because the direction fo the reaction is not set by default in bigg. We get all the models where the reaction is and the final direction will the one found in more than 75%

5./ Also extract xrefs

```
usage:
    padmet biggAPI_to_padmet --output=FILE [--pwy_file=FILE] [-v]

options:
    -h --help     Show help.
    --output=FILE    path to output, the padmet file.
    --pwy_file=FILE   add kegg pathways from pathways file, line:'pwy_id, pwy_name, x,
↪ rxn_id'.
    -v   print info.
```

padmet.utils.connection.biggAPI_to_padmet.**add_kegg_pwy**(*pwy_file*, *padmetRef*, *verbose=False*)
#TODO

padmet.utils.connection.biggAPI_to_padmet.**biggAPI_to_padmet**(*output*, *pwy_file=None*, *verbose=False*)
Extract BIGG database using the api. Create a padmet file. Escape reactions of biomass. Require internet access !

Allows to extract the bigg database from the API to create a padmet.

1./ Get all reactions universal id from http://bigg.ucsd.edu/api/v2/universal/reactions, escape reactions of biomass. 2./ Using async_list, extract all the informations for each reactions (compounds, stochio, name . . .) 3./ Need to use sleep time to avoid to lose the server access. 4./ Because the direction fo the reaction is not set by default in bigg. We get all the models where the reaction is and the final direction will the one found in more than 75% 5./ Also extract xrefs

Parameters

- **output** (*str*) – path to output, the padmet file.

- **pwy_file** (*str*) – path to pathway file, add kegg pathways, line:'pwy_id, pwy_name, x, rxn_id'.
- **verbose** (*bool*) – if True print information

padmet.utils.connection.biggAPI_to_padmet.**biggAPI_to_padmet_cli**(*command_args*)

padmet.utils.connection.biggAPI_to_padmet.**command_help**()
Show help for analysis command.

check_orthology_input

Description: Before running orthology based reconstruction it is necessary to check if the metabolic network and the proteome of the model organism use the same ids for genes (or at least more than a given cutoff). To only check this. Use the 2nd usage.

If the genes ids are not the same, it is necessary to use a dictionnary of genes ids associating the genes ids from the proteome to the genes ids from the metabolic network.

To create the correct proteome from the dictionnnary, use the 3nd usage Finnaly by using the 1st usage, it is possible to:

1/ Check model_faa and model_metabolic for a given cutoff

2/ if under the cutoff, convert model_faa to the correct one with dict_ids_file

3/ if still under, SystemExit()

```
usage:
    padmet check_orthology_input    --model_metabolic=FILE    --model_faa=FILE    [--
↪cutoff=FLOAT] [--dict_ids_file=FILE] --output=FILE    [-v]
    padmet check_orthology_input    --model_metabolic=FILE    --model_faa=FILE    [--
↪cutoff=FLOAT] [-v]
    padmet check_orthology_input    --model_faa=FILE    --dict_ids_file=FILE    --
↪output=FILE [-v]

option:
    -h --help    Show help.
    --model_metabolic=FILE    pathname to the metabolic network of the model (sbml).
    --model_faa=FILE    pathname to the proteome of the model (faa)
    --cutoff=FLOAT    cutoff [0:1] for comparing model_metabolic and model_faa.␣
↪[default: 0.70].
    --dict_ids_file=FILE    pathname to the dict associating genes ids from the model_
↪metabolic to the model_faa. line = gene_id_in_metabolic_network       gene_id_in_faa
    --output=FILE    output of get_valid_faa (a faa) or get_dict_ids (a dictionnary␣
↪of gene ids in tsv)
    -v   print info
```

padmet.utils.connection.check_orthology_input.**check_ids**(*model_metabolic*, *model_faa*, *cutoff*, *verbose=False*)
check if genes ids of model_metabolic = model_faa for a given cutoff faa genes ids are in the first line of each sequence: >GENE_ID .... metabolic netowkrs genes ids are in note section, GENE_ASSOCIATION: gene_id-1 or gene_id-2

Parameters

- **model_metabolic** (*str*) – path to sbml file
- **model_faa** (*str*) – path to fasta faa file

- **cutoff** (*int*) – cutoff genes ids from model found in faa
- **verbose** (*bool*) – verbose

Returns True if same ids, if verbose, print % of genes under cutoff

Return type bool

padmet.utils.connection.check_orthology_input.**check_orthology_input**(*model_metabolic*, *model_faa*, *dict_ids_file*, *output*, *verbose*, *cutoff*)

#TODO

padmet.utils.connection.check_orthology_input.**check_orthology_input_cli**(*command_args*)

padmet.utils.connection.check_orthology_input.**command_help**()

Show help for analysis command.

padmet.utils.connection.check_orthology_input.**get_valid_faa**(*model_faa*, *dict_ids_file*, *output*)

create a new faa from the model_faa by converting the gene id with the dict_ids dict_ids: line = origin_id new_gene_id, sep =

Parameters

- **model_faa** (*str*) – path to faa file
- **dict_ids_file** (*str*) – path to file containing link old to new ids
- **output** (*str*) – path to new faa file

enhanced_meneco_output

Description: The standard output of meneco return ids of reactions corresponding to the solution for gapfilling.

The ids are those from the sbml and so they are encoded.

This script extract the solution corresponding to the union of reactions "Computing union of reactions from all completion" Based on padmetRef return a file with more information for each reaction.

ex: RXN__45__5

RXN-5, common_name, ec-number, Formula (with id),Formula (with cname),Action,Comment Also, the output can be used as input of the script update_padmetSpec.py In the column Action: 'add' => To add the reaction, '' => to do nothing

Comment: the reason of adding the reaction (ex: added for gap-filling by meneco)

```
usage:
    padmet enhanced_meneco_output --meneco_output=FILE --padmetRef=FILE --output=FILE␣
↪[-v]

options:
    -h --help     Show help.
    --meneco_output=FILE    pathname of a meneco run' result
    --padmetRef=FILE    path to padmet file corresponding to the database of␣
↪reference (the repair network)
    --output=FILE    path to tsv output file
```

padmet.utils.connection.enhanced_meneco_output.**command_help**()
Show help for analysis command.

padmet.utils.connection.enhanced_meneco_output.**enhanced_meneco_output**(*meneco_output_file*, *padmetRef*, *output*, *verbose=False*)
The standard output of meneco return ids of reactions corresponding to the solution for gapfilling. The ids are those from the sbml and so they are encoded. This script extract the solution corresponding to the union of reactions "Computing union of reactions from all completion" Based on padmetRef return a file with more information for each reaction.

ex: RXN__45__5 RXN-5, common_name, ec-number, Formula (with id),Formula (with cname),Action,Comment Also, the output can be used as input for manual_curation In the column Action: 'add' => To add the reaction, '' => to do nothing Comment: the reason of adding the reaction (ex: added for gap-filling by meneco)

Parameters

- **meneco_output_file** (*str*) – pathname of a meneco run' result
- **padmetRef** (*padmet.padmetRef*) – path to padmet file corresponding to the database of reference (the repair network)
- **output** (*str*) – path to tsv output file
- **verbose** (*bool*) – if True print information

padmet.utils.connection.enhanced_meneco_output.**enhanced_meneco_output_cli**(*command_args*)

extract_orthofinder

Description: After running orthofinder on n fasta file, read the output file 'Orthogroups.tsv'

Require a folder 'orthology_based_folder' with this archi:

|– **model_a** – model_a.sbml

|– **model_b** –model_b.sbml

And the name of the studied organism 'study_id'

1. Read the orthogroups file, extract orthogroups in dict 'all_orthogroups', and all org names
2. In orthology folder search for sbml files 'extension = .sbml'
3. For each models regroup all information in a dict dict_data:

   {'study_id': study_id, 'model_id' : model_id, 'sbml_template': path to sbml of model', 'output': path to the output sbml, 'verbose': bool, if true print information }

   The output is by default: output_orthofinder_from_'model_id'.sbml
4. Store all previous dict_data in a list all_dict_data
5. iter on dict from all_dict_data and use function dict_data_to_sbml

Use a dict of data dict_data and dict of orthogroups dict_orthogroup to create sbml files.

dict_data and dict_orthogroup are obtained with fun orthofinder_to_sbml

6./ Read dict_orthogroups and check if model associated to dict_data and study org share orthologue

7./ Read sbml of model, parse all reactions and get genes associated to reaction.

8./ For each reactions:

Parse genes associated to sub part (ex: (gene-a and gene-b) or gene-c) = [(gene-a,gene-b), gene-c]

Check if study org have orthologue with at least one sub part (gene-a, gene-b) or gene-c

if yes: add the reaction to the new sbml and change genes ids by study org genes ids

Create the new sbml file.

```
usage:
    padmet extract_orthofinder --sbml=FILE/DIR --orthologues=DIR --study_id=STR --
↪output=DIR [--workflow=STR] [-v]
    padmet extract_orthofinder --sbml=DIR --orthogroups=FILE --study_id=STR --
↪output=DIR [--workflow=STR] [-v]

option:
    -h --help    Show help.
    --sbml=DIR   Folder with sub folder named as models name within sbml file name as␣
↪model_name.sbml
    --orthogroups=FILE   Output file of Orthofinder run Orthogroups.tsv
    --orthologues=DIR   Output directory of Orthofinder run Orthologues
    --study_id=ID   name of the studied organism
    --workflow=ID   worklow id in ['aureme','aucome']. specific run architecture␣
↪where to search sbml files
   --output=DIR   folder where to create all sbml output files
    -v   print info
```

padmet.utils.connection.extract_orthofinder.**command_help**()

Show help for analysis command.

padmet.utils.connection.extract_orthofinder.**dict_data_to_sbml**(*dict_data*, *dict_orthogroups=None*, *dict_orthologues=None*, *strict_match=True*)

Use a dict of data dict_data and dict of orthogroups dict_orthogroup to create sbml files. dict_data and dict_orthogroup are obtained with fun orthofinder_to_sbml 1./ Read dict_orthogroups and check if model associated to dict_data and study org share orthologue 2./ Read sbml of model, parse all reactions and get genes associated to reaction. 3./ For each reactions:

Parse genes associated to sub part (ex: (gene-a and gene-b) or gene-c) = [(gene-a,gene-b), gene-c] Check if study org have orthologue with at least one sub part (gene-a, gene-b) or gene-c if yes: add the reaction to the new sbml and change genes ids by study org genes ids

4./ Create the new sbml file.

Parameters

- **dict_data** (*dict*) – {'study_id': study_id, 'model_id' : model_id, 'sbml_template': path to sbml of model', 'output': path to the output sbml, 'verbose': bool, if true print information }
- **dict_orthogroup** (*dict*) – k=orthogroup_id, v = {k = name, v = set of genes}
- **verbose** (*bool*) – if True print information

padmet.utils.connection.extract_orthofinder.**extract_orthofinder_cli**(*command_args*)

padmet.utils.connection.extract_orthofinder.**get_sbml_files**(*sbml*, *workflow=None*, *verbose=False*)

#TODO

padmet.utils.connection.extract_orthofinder.**orthogroups_to_sbml**(*orthogroups_file*, *all_model_sbml*, *output_folder*, *study_id*, *verbose=False*)

After running orthofinder on n fasta file, read the output file 'Orthogroups.tsv' Require a folder 'orthology_based_folder' with this archi: model_a

model_a.sbml

model_b model_b.sbml

And the name of the studied organism 'study_id' 1. Read the orthogroups file, extract orthogroups in dict 'all_orthogroups', and all org names 2. In orthology folder search for sbml files 'extension = .sbml' 3. For each models regroup all information in a dict dict_data:

{'study_id': study_id, 'model_id' : model_id, 'sbml_template': path to sbml of model', 'output': path to the output sbml, 'verbose': bool, if true print information } The output is by default: **output_orthofinder_from_**'model_id'.sbml

4. Store all previous dict_data in a list all_dict_data

5. iter on dict from all_dict_data and use function dict_data_to_sbml This function will create a sbml from each model and conserve only reactions associated to ortholog genes For more information read the doc of func dict_data_to_sbml

Parameters

- **orthogroups_file** (*str*) – path of Orthofinder output file 'Orthogroups.tsv'
- **orthology_based_folder** (*str*) – path of folder with model's sbml
- **output** (*str*) – pathname of the output folder of all sbml extracted
- **study_id** (*str*) – name of the studied organism
- **verbose** (*bool*) – if True print information

padmet.utils.connection.extract_orthofinder.**orthologue_to_sbml**(*orthologue_folder*, *all_model_sbml*, *output_folder*, *study_id*, *verbose=False*)

After running orthofinder on n fasta file, read the output files in 'Orthologues' Require a folder 'orthology_based_folder' with this archi: model_a

model_a.sbml

model_b model_b.sbml

And the name of the studied organism 'study_id' 1. Read the orthogroups file, extract orthogroups in dict 'all_orthogroups', and all org names 2. In orthology folder search for sbml files 'extension = .sbml' 3. For each models regroup all information in a dict dict_data:

{'study_id': study_id, 'model_id' : model_id, 'sbml_template': path to sbml of model', 'output': path to the output sbml, 'verbose': bool, if true print information } The output is by default: **output_orthofinder_from_**'model_id'.sbml

4. Store all previous dict_data in a list all_dict_data

5. iter on dict from all_dict_data and use function dict_data_to_sbml This function will create a sbml from each model and conserve only reactions associated to ortholog genes For more information read the doc of func dict_data_to_sbml

Parameters

- **orthologue_folder** (*str*) – path of Orthofinder output folder 'Orthologues'
- **orthology_based_folder** (*str*) – path of folder with model's sbml
- **output** (*str*) – pathname of the output folder of all sbml extracted
- **study_id** (*str*) – name of the studied organism
- **verbose** (*bool*) – if True print information

extract_rxn_with_gene_assoc

Description: From a given sbml file, create a sbml with only the reactions associated to a gene.

Need for a reaction, in section 'note', 'GENE_ASSOCIATION': ....

```
usage:
    padmet extract_rxn_with_gene_assoc --sbml=FILE --output=FILE [-v]

options:
    -h --help     Show help.
    --sbml=FILE    path to the sbml file
    --output=FILE    path to the sbml output (with only rxn with genes assoc)
    -v   print info
```

padmet.utils.connection.extract_rxn_with_gene_assoc.**command_help**()
Show help for analysis command.

padmet.utils.connection.extract_rxn_with_gene_assoc.**extract_rxn_with_gene_assoc**(*sbml*, *output*, *verbose=False*)
From a given sbml document, create a sbml with only the reactions associated to a gene. Need for a reaction, in section 'note', 'GENE_ASSOCIATION': ....

Parameters

- **sbml_file** (*libsbml.document*) – sbml document
- **output** (*str*) – pathname of the output sbml

padmet.utils.connection.extract_rxn_with_gene_assoc.**extract_rxn_with_gene_assoc_cli**(*command*

gbk_to_faa

Description: convert GBK to FAA with Bio package

```
usage:
    padmet gbk_to_faa    --gbk=FILE --output=FILE [--qualifier=STR] [-v]
```

(continues on next page)

(continued from previous page)

```
option:
    -h --help    Show help.
    --gbk=FILE    path to the gbk file.
    --output=FILE    path to the output, a FAA file.
    --qualifier=STR    the qualifier of the gene id [default: locus_tag].
    -v   print info
```

padmet.utils.connection.gbk_to_faa.**command_help**()
Show help for analysis command.

padmet.utils.connection.gbk_to_faa.**gbk_to_faa**(*gbk_file*, *output*, *qualifier='locus_tag'*, *verbose=True*)
convert GBK to FAA with Bio package

Parameters

- **gbk_file** (*str*) – path to the gbk file
- **output** (*str*) – path to the output, a FAA file
- **qualifier** (*str*) – he qualifier of the gene id
- **verbose** (*bool*) – if True print information

padmet.utils.connection.gbk_to_faa.**gbk_to_faa_cli**(*command_args*)

gene_to_targets

Description: From a list of genes, get from the linked reactions the list of products.

R1 is linked to G1, R1 produces M1 and M2. output: M1,M2. Takes into account reversibility

```
usage:
    padmet gene_to_targets --padmetSpec=FILE --genes=FILE --output=FILE [-v]

option:
    -h --help     Show help
    --padmetSpec=FILE    path to the padmet file
    --genes=FILE   path to the file containing gene ids, one id by line
    --output=FILE    path to the output file containing all tagerts which can by␣
↪produced by all reactions associated to the given genes
    -v   print info
```

padmet.utils.connection.gene_to_targets.**command_help**()
Show help for analysis command.

padmet.utils.connection.gene_to_targets.**gene_to_targets**(*padmet*, *genes_file*, *output*, *verbose=False*)
From a list of genes, get from the linked reactions the list of products. R1 is linked to G1, R1 produces M1 and M2. output: M1,M2. Takes into account reversibility

Parameters

- **padmet** (*padmet.classes.PadmetSpec*) – padmet to explore
- **genes_file** (*str*) – path of genes file, 1 gene id by line
- **output** (*str*) – pathname of the output file
- **verbose** (*bool*) – if True print information

padmet.utils.connection.gene_to_targets.**gene_to_targets_cli**(*command_args*)

modelSeed_to_padmet

Description: From modelSeed reactions and pathways files creates a padmet file.

```
usage:
    padmet modelSeed_to_padmet --output=FILE --rxn_file=FILE --pwy_file=FILE [-v]

options:
    -h --help     Show help.
    --output=FILE    path of the padmet file to create
    --rxn_file=FILE   path to json file of modelSeed reactions
    --pwy_file=FILE   path to pathway reactions association from modelSeed
    -v   print info.
```

padmet.utils.connection.modelSeed_to_padmet.**add_kegg_pwy**(*pwy_file*, *padmetRef*, *verbose=False*)

#TODO

padmet.utils.connection.modelSeed_to_padmet.**command_help**()

Show help for analysis command.

padmet.utils.connection.modelSeed_to_padmet.**modelSeed_to_padmet**(*rxn_file*, *pwy_file*, *output*, *verbose=False*)

#TODO

padmet.utils.connection.modelSeed_to_padmet.**modelSeed_to_padmet_cli**(*command_args*)

padmet_to_asp

Description: Convert PADMet to ASP following these predicats: common_name({reaction_id or enzyme_id or pathway_id or compound_id} , common_name) direction(reaction_id, reaction_direction). reaction_direction in[LEFT-TO-RIGHT,REVERSIBLE] ec_number(reaction_id, ec(x,x,x)). catalysed_by(reaction_id, enzyme_id). uniprotID(enzyme_id, uniprot_id). #if has has_xref and db = "UNIPROT" in_pathway(reaction_id, pathway_id). reactant(reaction_id, compound_id, stoechio_value). product(reaction_id, compound_id, stoechio_value). is_a(compound_id, class_id). is_a(pathway_id, pathway_id).

```
usage:
    padmet padmet_to_asp --padmet=FILE --output=FILE [-v]

option:
    -h --help     Show help.
    --padmet=FILE    path to padmet file to convert.
    --output=FILE    path to output file in lp format.
    -v    print info.
```

padmet.utils.connection.padmet_to_asp.**asp_synt**(*pred*, *list_args*)

create a predicat for asp

example: asp_synt("direction",["R1","REVERSIBLE"]) => "direction('R1','reversible')."

Parameters

- **pred** (*str*) – the predicat
- **list_args** (*list*) – list of atoms to put in the predicat

Returns the predicat 'pred(''list_args[0]'','list_args[1]',...,'list_args[n]').'

Return type str

padmet.utils.connection.padmet_to_asp.**command_help**()
Show help for analysis command.

padmet.utils.connection.padmet_to_asp.**padmet_to_asp**(*padmet_file*, *output*, *verbose=False*)
Convert PADMet to ASP following these predicats: common_name({reaction_id or enzyme_id or pathway_id or compound_id} , common_name) direction(reaction_id, reaction_direction). reaction_direction in[LEFT-TO-RIGHT,REVERSIBLE] ec_number(reaction_id, ec(x,x,x)). catalysed_by(reaction_id, enzyme_id). uniprotID(enzyme_id, uniprot_id). #if has has_xref and db = "UNIPROT" in_pathway(reaction_id, pathway_id). reactant(reaction_id, compound_id, stoechio_value). product(reaction_id, compound_id, stoechio_value). is_a(compound_id, class_id). is_a(pathway_id, pathway_id).

Parameters

- **padmet_file** (*str*) – the path to padmet file to convert
- **output** (*str*) – the path to the output to create
- **verbose** (*bool*) – print informations

padmet.utils.connection.padmet_to_asp.**padmet_to_asp_cli**(*command_args*)

padmet_to_matrix

Description: Create a stoichiometry matrix from a padmet file.

The columns represent the reactions and rows represent metabolites.

S[i,j] contains the quantity of metabolite 'i' produced (negative for consumed) by reaction 'j'.

```
usage:
    padmet padmet_to_matrix --padmet=FILE --output=FILE

option:
    -h --help    Show help.
    --padmet=FILE    path to the padmet file to convert.
    --output=FILE    path to the output file, col: rxn, row: metabo, sep = "        ".
```

padmet.utils.connection.padmet_to_matrix.**command_help**()
Show help for analysis command.

padmet.utils.connection.padmet_to_matrix.**padmet_to_matrix**(*padmet*, *output*)
Create a stoichiometry matrix from a padmet file. The columns represent the reactions and rows represent metabolites. S[i,j] contains the quantity of metabolite 'i' produced (negative for consumed) by reaction 'j'.

Parameters

- **padmet** (*padmet.PadmetSpec*) – padmet instance
- **output** – path to the output file, col: rxn, row: metabo, sep = " "

padmet.utils.connection.padmet_to_matrix.**padmet_to_matrix_cli**(*command_args*)

padmet_to_padmet

Description: Allows to merge 1-n padmet. 1./ Update the 'init_padmet' with the 'to_add' padmet(s). to_add can be a file or a folder with only padmet files to add.

```
usage:
    padmet padmet_to_padmet --to_add=FILE/DIR --output=FILE  [-v]

options:
    -h --help     Show help.
    --to_add=FILE/DIR    path to the padmet file to add (sep: ,) or path to folder of␣
↪padmet files.
    --output=FILE   path to the new padmet file
    -v   print info
```

padmet.utils.connection.padmet_to_padmet.**command_help**()
Show help for analysis command.

padmet.utils.connection.padmet_to_padmet.**padmet_to_padmet**(*to_add*, *output=None*, *verbose=False*)
Create a padmet by merging multiple other padmet files.

Parameters

- **to_add** (*dir or str*) – padmet directory or string with multiple padmet paths separated by ','
- **output** – path to the output file
- **verbose** (*bool*) – verbose level of script

Returns **padmet_init** – padmet created from emrging of the other padmet

Return type *PadmetSpec*

padmet.utils.connection.padmet_to_padmet.**padmet_to_padmet_cli**(*command_args*)

padmet_to_tsv

Description: convert a padmet representing a database (padmetRef) and/or a padmet representing a model (padmetSpec) to tsv files for askomics.

1./ Folder creation given the output directory. Create this directory if required and create a folder padmetRef filename and/or padmetSpec filename

2./

2.1/ For padmetRef:

2.1.a/ Nodes get all reactions nodes => extract data from misc with extract_nodes(rxn_nodes, "reaction", "../rxn.tsv")

get all compounds nodes => extract data from misc with extract_nodes(cpd_nodes, "compounds", "../cpd.tsv")

get all pathways nodes => extract data from misc with extract_nodes(pwy_nodes, "pathway", "../pwy.tsv")

get all xrefs nodes => extract data from misc with extract_nodes(xref_nodes, "xref", "../xref.tsv")

2.1.b/ Relations for each rxn in rxn nodes:

get all rlt consumes/produces => create list of data with extract_rxn_cpd(rxn_cpd_rlt)
fieldnames = "rxn_cpd","concerns@reaction","consumes@compound","produces@compound","stoichiometry",

get all rlt is_in_pathway => create list of data with extract_rxn_pwy(rxn_pwy_rlt)
fieldnames = "rxn_pwy","concerns@reaction","in_pwy@pathway"

get all rlt has_xref => create list of data with extract_entity_xref(rxn_xref_rlt)

for each cpd in cpd nodes:

get all rlt has_xref => update previous list of data with extract_entity_xref(cpd_xref_rlt)
fieldnames = "entity_xref","concerns@reaction","concerns@compound","has_xref@xref"

```
usage:
    padmet padmet_to_tsv --padmetSpec=FILE [--padmetRef=FILE] --output_dir=DIR [-v]
    padmet padmet_to_tsv --padmetRef=FILE [--padmetSpec=FILE] --output_dir=DIR [-v]

options:
    -h --help     Show help.
    --padmetSpec=FILE    path of the padmet representing the network to convert
    --padmetRef=FILE    path of the padmet representing the database
    --output_dir=DIR
    -v
```

padmet.utils.connection.padmet_to_tsv.**command_help**()
Show help for analysis command.

padmet.utils.connection.padmet_to_tsv.**entity_xref_file**(*data*, *output*)
#TODO

padmet.utils.connection.padmet_to_tsv.**extract_entity_xref**(*entity_xref_rlt*, *padmet*)
#TODO

padmet.utils.connection.padmet_to_tsv.**extract_nodes**(*padmet*, *nodes*, *entity_id*, *output*, *opt_col={}*)
for n nodes in nodes. for each node.misc = {A:['x'],B:['y','z']} create a file with line = [node.id,A[0],B[0]],[node.id,"",B[1]] the order is defined in fieldnames. merge common name and synonyms in 'name'

#TODO

padmet.utils.connection.padmet_to_tsv.**extract_pwy**(*padmet*)
from padmet return a dict, k = pwy_id, v = set of rxn_id in pwy

#TODO

padmet.utils.connection.padmet_to_tsv.**extract_rxn_cpd**(*rxn_cpd_rlt*)
for rlt in rxn_cpd_rlt, append in data: [rxn_id,cpd_id(consumed),'',stoich,compartment] and/or [rxn_id,'',cpd_id(produced),stoich,compartment]. The value in index 0 is a merge of all data to create a unique relation id

#TODO

padmet.utils.connection.padmet_to_tsv.**extract_rxn_gene**(*rxn_gene_rlt*)
#TODO

padmet.utils.connection.padmet_to_tsv.**extract_rxn_pwy**(*rxn_pwy_rlt*)
for rlt in rxn_pwy_rlt, append in data: [rxn_id,pwy_id]. The value in index 0 is a merge of all data to create a unique relation id

#TODO

padmet.utils.connection.padmet_to_tsv.**extract_rxn_rec**(*rxn_rec_rlt*)
#TODO

padmet.utils.connection.padmet_to_tsv.**padmet_to_tsv**(*padmetSpec_file*, *padmetRef_file*, *output_dir*, *verbose=False*)

#TODO

padmet.utils.connection.padmet_to_tsv.**padmet_to_tsv_cli**(*command_args*)

padmet.utils.connection.padmet_to_tsv.**pwy_rate**(*padmetRef*, *padmetSpec*, *metabolic_network*, *output*)

pwy rate in padmetSpec is calculated based on padmetRef

#TODO

padmet.utils.connection.padmet_to_tsv.**rxn_cpd_file**(*data*, *output*)

from data obtained with extract_rxn_cpd(), create file rxn_cpd

#TODO

padmet.utils.connection.padmet_to_tsv.**rxn_gene_file**(*data*, *output*)

#TODO

padmet.utils.connection.padmet_to_tsv.**rxn_pwy_file**(*data*, *output*)

#TODO

padmet.utils.connection.padmet_to_tsv.**rxn_rec_file**(*data*, *output*)

#TODO

pgdb_to_padmet

Description:

Read a PGDB folder (from BIOCYC/PATHWAYTOOLS) and create a padmet. 1./ To create a padmet without any genes information extracted use the first usage with:

pgdb: path to pgdb folder output: path to the padmet to create version: to specify the version of the pgdb (20.0, 22.0) db: to sepcify the name of the database (METACYC, ECOCYC, ...) enhance: to also read the file metabolic-reaction.xml and add the to the padmet

2./ To create a padmet and add only reactions from pgdb if they are in padmetRef specifie. Copy information of the reaction not from the pgdb but from the padmetRef. This allow to uniform reaction to the same version of metacyc represented in the padmetRef For example, in some case 2 pgdb from different version can contain different information for a same reaction,pathway... In this case use:

padmetRef: path to the padmet of reference

3./ To create a padmet wth genes information extracted use: extract-gene

3.1/ To remove from the final padmet all reactions without genes associated use: no-orphan

4./ To read the metabolic-reaction.xml file, a sbml with some missing reactions in PGDB use: enhance

For more information of the parsing process read information below.

classes.dat: For each class: create new node / class = class UNIQUE-ID (1) => node.id = UNIQUE-ID COMMON-NAME (0-n) => node.Misc['COMMON-NAME'] = COMMON-NAME TYPES (0-n) => for each, check or create new node class, create rlt (node is_a_class types) SYNONYMS (0-n) => for each, create new node name, create rlt (node has_name synonyms)

compounds.dat: for each compound: create new node / class = compound UNIQUE-ID (1) => node.id = UNIQUE-ID COMMON-NAME (0-n) => node.Misc['COMMON-NAME'] = COMMON-NAME INCHI-KEY (0-1) {InChIKey=XXX} => node.misc['INCHI_KEY': XXX] MOLECULAR-WEIGHT (0-1) => node.misc()['MOLECULAR_WEIGHT'] = MOLECULAR-WEIGHT SMILES (0-1)

=> node.misc()['SMILES'] = SMILES TYPES (0-n) => for each, check or create new node class, create rlt (node is_a_class types) SYNONYMS (0-n) => for each, create new node name, create rlt (node has_name name) DBLINKS (0-n) {(db "id" ...)} => for each, create new node xref, create rlt (node has_xref xref)

proteins.dat: for each protein: create new node / class = protein UNIQUE-ID (1) => node.id = UNIQUE-ID COMMON-NAME (0-n) => node.Misc['COMMON-NAME'] = COMMON-NAME INCHI-KEY (0-1) {InChIKey=XXX} => node.misc['INCHI_KEY': XXX] MOLECULAR-WEIGHT (0-1) => node.misc()['MOLECULAR_WEIGHT'] = MOLECULAR-WEIGHT SMILES (0-1) => node.misc()['SMILES'] = SMILES TYPES (0-n) => for each, check or create new node class, create rlt (node is_a_class types) SYNONYMS (0-n) => for each, create new node name, create rlt (node has_name name) DBLINKS (0-n) {(db "id" ...)} => for each, create new node xref, create rlt (node has_xref xref) SPECIES (0-1) => for each, check or create new node class, create rlt (node is_in_species class)

reactions.dat: for each reaction: create new node / class = reaction + node.misc()["DIRECTION"] = "UNKNOWN" by default UNIQUE-ID (1) => node.id = UNIQUE-ID COMMON-NAME (0-n) => node.Misc['COMMON-NAME'] = COMMON-NAME EC-NUMBER (0-n) => node.Misc['EC-NUMBER'] = EC-NUMBER REACTION-DIRECTION (0-1) => node.Misc['DIRECTION'] = reaction-direction, if REVERSIBLE, else: LEFT-TO-RIGHT RXN-LOCATIONS (0,n) => node.misc['COMPARTMENT'] = rxn-location TYPES (0-n) => check or create new node class, create rlt (node.id is_a_class types's_node.id) DBLINKS (0-n) {(db "id" ...)} => create new node xref, create rlt (node has_xref xref's_node.id) SYNONYMS (0-n) => create new node name, create rlt (node has_name name's_node.id) – for LEFT and RIGHT, also check 2 next lines if info about 'coefficient' or 'compartment' defaut value: coefficient/stoichiometry = 1, compartment = unknown also check if the direction is 'RIGHT-TO-LEFT', if yes, inverse consumes and produces relations then change direction to 'LEFT-TO-RIGHT' LEFT (1-n) => create rlt (node.id consumes left's_node.id) RIGHT (1-n) => create rlt (node.id produces right's_node.id)

enzrxns.dat: for each association enzyme/reaction: create new rlt / type = catalyses ENZYME (1) => stock enzyme as 'enzyme catalyses' REACTION (1-n) => for each reaction after, create relation 'enzyme catalyses reaction'

pathways.dat: for each pathway: create new node / class = pathway UNIQUE-ID (1) => node._id = UNIQUE-ID TYPES (0-n) => check or create new node class, create rlt (node is_a_class types) COMMON-NAME (0-n) => node.Misc['COMMON-NAME'] = COMMON-NAME DBLINKS (0-n) {(db "id" ...)} => create new node xref, create rlt (node has_xref xref) SYNONYMS (0-n) => create new node name, create rlt (node has_name name) IN-PATHWAY (0-n) => check or create new node pathway, create rlt (node is_in_pathway name) REACTION-LIST (0-n) => check or create new node pathway, create rlt (node is_in_pathway name)

```
usage:
    padmet pgdb_to_padmet --pgdb=DIR --output=FILE [--version=V] [--db=ID] [--
↪padmetRef=FILE] [--source=STR] [-v] [--enhance]
    padmet pgdb_to_padmet --pgdb=DIR --output=FILE --extract-gene [--no-orphan] [--
↪keep-self-rxn] [--version=V] [--db=ID] [--padmetRef=FILE] [--source=STR] [-v] [--
↪enhance]

options:
    -h --help     Show help.
    --version=V    Xcyc version [default: N.A].
    --db=ID    Biocyc database corresponding to the pgdb (metacyc, ecocyc, ...)␣
↪[default: N.A].
    --output=FILE    padmet file corresponding to the DB.
    --pgdb=DIR    directory containg all the .dat files of metacyc (data).
    --padmetRef=FILE    padmet of reference.
    --source=STR    Tag associated to the source of the reactions, used to ensure␣
↪traceability [default: GENOME].
```

(continues on next page)

(continued from previous page)

```
    --enhance    use the metabolic-reactions.xml file to enhance the database.
    --extract-gene    extract genes from genes_file (use if its a specie's pgdb, if␣
↪metacyc, do not use).
    --no-orhpan    remove reactions without gene associaiton (use if its a specie's␣
↪pgdb, if metacyc, do not use).
    --keep-self-rxn    remove reactions with no reactants (use if its a specie's pgdb,
↪ if metacyc, do not use).
    -v   print info.
```

padmet.utils.connection.pgdb_to_padmet.**classes_parser**(*filePath*, *padmet*, *verbose=False*)

from class.dat: get for each class, the UNIQUE-ID, COMMON-NAME, TYPES, SYNONYMS, DBLINKS Create a class node with node.id = UNIQUE-ID, node.misc = {COMMON-NAME:[COMMON-NAMES]} - For each types: A type is in fact a class. this information is stocked in padmet as: is_a_class relation btw a node and a class_node check if the type is already in the padmet if not create a new class_node (var: subClass) with subClass_node.id = type Create a relation current node is_a_class type - For each Synonyms: this information is stocked in padmet as: has_name relation btw a node and a name_node create a new name_node with name_node.id = class_id+"_names" and name_node.misc = {LABEL:[synonyms]} Create a relation current node has_name name_node.id - For each DBLINKS: DBLINKS is parsed with regex_xref to get the db and the id this information is stocked in padmet as: has_xref relation btw a node and a xref_node create a new xref_node with xref_node.id = class_id+"_xrefs" and xref_node.misc = {db:[id]} Create a relation current node has_xref xref_node.id

Parameters

- **filePath** (*str*) – path to classes.dat
- **padmet** (*padmet.PadmetRef*) – padmet instance
- **verbose** (*bool*) – if True print information

padmet.utils.connection.pgdb_to_padmet.**command_help**()

Show help for analysis command.

padmet.utils.connection.pgdb_to_padmet.**compounds_parser**(*filePath*, *padmet*, *verbose=False*)

Parameters

- **filePath** (*str*) – path to compounds.dat
- **padmet** (*padmet.PadmetRef*) – padmet instance
- **verbose** (*bool*) – if True print information

padmet.utils.connection.pgdb_to_padmet.**enhance_db**(*metabolic_reactions*, *padmet*, *with_genes*, *verbose=False*)

Parse sbml metabolic_reactions and add reactions in padmet if with_genes: add also genes information

Parameters

- **metabolic_reactions** (*str*) – path to sbml metabolic-reactions.xml
- **padmet** (*padmet.PadmetRef*) – padmet instance
- **with_genes** (*bool*) – if true alos add genes information.

Returns padmet instance with pgdb within pgdb + metabolic-reactions.xml data

Return type padmet.padmetRef

padmet.utils.connection.pgdb_to_padmet.**enzrxns_parser**(*filePath*, *padmet*, *dict_protein_gene_id*, *source*, *verbose=False*)

Parameters

- **filePath** (*str*) – path to enzrxns.dat
- **padmet** (*padmet.PadmetRef*) – padmet instance
- **verbose** (*bool*) – if True print information

padmet.utils.connection.pgdb_to_padmet.**from_pgdb_to_padmet**(*pgdb_folder*, *db='MetaCyc'*, *version='NA'*, *source='GENOME'*, *extract_gene=False*, *no_orphan=False*, *no_self_producing_rxn=True*, *enhanced_db=False*, *padmetRef_file=None*, *verbose=False*, *output_file=None*)

Parameters

- **pgdb_folder** (*str*) – path to pgdb
- **db** (*str*) – pgdb name, default is 'NA'
- **version** (*str*) – pgdb version, default is 'NA'
- **source** (*str*) – tag reactions for traceability, default is 'GENOME'
- **extract_gene** (*bool*) – if true extract genes information
- **no_orphan** (*bool*) – if true, remove reactions without genes associated
- **no_self_producing_rxn** (*bool*) – if true, remove reactions with no reactants (auto-producing reactions)
- **enhanced_db** (*bool*) – if true, read metabolix-reactions.xml sbml file and add information in final padmet
- **padmetRef_file** (*str*) – path to padmetRef corresponding to metacyc in padmet format
- **verbose** (*bool*) – if True print information
- **output_file** (*str*) – pathname of padmet output file

Returns padmet instance with pgdb within pgdb data

Return type padmet.padmetRef

padmet.utils.connection.pgdb_to_padmet.**genes_parser**(*filePath*, *padmet*, *verbose=False*)

Parameters

- **filePath** (*str*) – path to genes.dat
- **padmet** (*padmet.PadmetRef*) – padmet instance
- **verbose** (*bool*) – if True print information

padmet.utils.connection.pgdb_to_padmet.**map_gene_id**(*dict_protein_gene_id*, *map_gene_ids*)

Map gene ID created by Pathway Tools with gene ID from the data. Automatically Pathway Tools uppercased all the letter in gene ID. So we need to do this mapping to retrieve the unuppercased gene ID.

padmet.utils.connection.pgdb_to_padmet.**pathways_parser**(*filePath*, *padmet*, *verbose=False*)

Parameters

- **filePath** (*str*) – path to pathways.dat
- **padmet** (*padmet.PadmetRef*) – padmet instance
- **verbose** (*bool*) – if True print information

padmet.utils.connection.pgdb_to_padmet.**pgdb_to_padmet_cli**(*command_args*)

padmet.utils.connection.pgdb_to_padmet.**proteins_parser**(*filePath*, *padmet*, *compounds*, *rnas*, *id_classes*, *verbose=False*)

Parameters

- **filePath** (*str*) – path to proteins.dat
- **padmet** (*padmet.PadmetRef*) – padmet instance
- **compounds** (*list*) – list of known compounds in the species
- **verbose** (*bool*) – if True print information

padmet.utils.connection.pgdb_to_padmet.**reactions_parser**(*filePath*, *padmet*, *extract_gene*, *source*, *verbose=False*)

from reaction.dat: get for each reaction, the UNIQUE-ID, COMMON-NAME, TYPES, SYNONYMS, DBLINKS Create a reaction node with node.id = UNIQUE-ID, node.misc = {COMMON-NAME:[COMMON-NAMES]} - For each types: A type is in fact a class. this information is stocked in padmet as: is_a_class relation btw a node and a class_node check if the type is already in the padmet if not create a new class_node (var: subClass) with subClass_node.id = type Create a relation current node is_a_class type - For each Synonyms: this information is stocked in padmet as: has_name relation btw a node and a name_node create a new name_node with name_node.id = reaction_id+"_names" name_node.misc = {LABEL:[synonyms]} Create a relation current node has_name name_node.id - For each DBLINKS: DBLINKS is parsed with regex_xref to get the db and the id this information is stocked in padmet as: has_xref relation btw a node and a xref_node create a new xref_node with xref_node.id = reaction_id+"_xrefs" and xref_node.misc = {db:[id]} Create a relation current node has_xref xref_node.id

Parameters

- **filePath** (*str*) – path to reactions.dat
- **padmet** (*padmet.PadmetRef*) – padmet instance
- **verbose** (*bool*) – if True print information

padmet.utils.connection.pgdb_to_padmet.**rnas_parser**(*filePath*, *padmet*, *verbose=False*)

Parameters

- **filePath** (*str*) – path to compounds.dat
- **padmet** (*padmet.PadmetRef*) – padmet instance
- **verbose** (*bool*) – if True print information

sbmlGenerator

Description: The module sbmlGenerator contains functions to generate sbml files from padmet and txt usign the libsbml package

```
usage:
    padmet sbmlGenerator --padmet=FILE --output=FILE --sbml_lvl=STR [--model_id=STR]␣
↪[--obj_fct=STR] [--mnx_chem_prop=FILE] [--mnx_chem_xref=FILE] [-v]
    padmet sbmlGenerator --padmet=FILE --output=FILE [--init_source=STR] [-v]
    padmet sbmlGenerator --compound=FILE --output=FILE [--padmetRef=FILE] [-v]
    padmet sbmlGenerator --reaction=FILE --output=FILE --padmetRef=FILE [-v]

option:
    -h --help    Show help.
    --padmet=FILE    path of the padmet file to convert into sbml
    --output=FILE    path of the sbml file to generate.
    --mnx_chem_prop=FILE    path of the MNX chemical compounds properties.
    --mnx_chem_xref=FILE    path of the mnx dict of chemical compounds id mapping.
    --reaction=FILE    path of file of reactions ids, one by line to convert to sbml.
    --compound=FILE    path of file of compounds ids, one by line to convert to sbml.
    --init_source=STR    Select the reactions of padmet to convert on sbml based on␣
↪the source of the reactions, check relations rxn has_reconstructionData.
    --sbml_lvl=STR    sbml level of output. [default 3]
    --obj_fct=STR    id of the reaction objective.
    -v   print info.
```

padmet.utils.connection.sbmlGenerator.**add_ga**(*rId_encoded*, *all_ga_subsets*)
if list_ga len == 1: only 1 list of gene: if len of this list is 1: just add gene, else create OR structure else: create OR structure, then for each list of gene for each ga in list_ga: if len == 1: if the only ga len == 1: just add gene, else create OR structure elif len > 1: create AND structure, then for each GA if len GA == 1: just add gene, else create OR structure if no suppdata, if linked_genes: if len linked_genes == 1: just add gene, else create OR structure #TODO

padmet.utils.connection.sbmlGenerator.**check**(*value*, *message*)
If 'value' is None, prints an error message constructed using 'message' and then exits with status code 1. If 'value' is an integer, it assumes it is a libSBML return status code. If the code value is LIBSBML_OPERATION_SUCCESS, returns without further action; if it is not, prints an error message constructed using 'message' along with text from libSBML explaining the meaning of the code, and exits with status code 1.

padmet.utils.connection.sbmlGenerator.**command_help**()
Show help for analysis command.

padmet.utils.connection.sbmlGenerator.**compound_to_sbml**(*species_compart*, *output*, *verbose=False*)
convert a list of compounds to sbml format if compart_name is not None, then the compounds id will by: M_originalID_compart_name if verbose and specified padmetRef and/or padmetSpec: will check if compounds are in one of the padmet files Ids are encoded for sbml using functions sbmlPlugin.convert_to_coded_id

Parameters

- **species_file** (*str*) – pathname to the file containing the compounds ids and the compart, line = cpd-id compart.
- **output** (*str*) – pathname to the sbml file to create
- **verbose** (*bool*) – print informations

padmet.utils.connection.sbmlGenerator.**create_annotation**(*inchi*, *ref_id*)
dict_data, k = url, v = id #TODO

padmet.utils.connection.sbmlGenerator.**create_note**(*dict_data*)
#TODO

padmet.utils.connection.sbmlGenerator.**from_init_source**(*padmet_file*, *init_source*, *output*, *verbose=False*)
#TODO

padmet.utils.connection.sbmlGenerator.**padmet_to_sbml**(*padmet*, *output*, *model_id=None*, *obj_fct=None*, *sbml_lvl=3*, *mnx_chem_prop=None*, *mnx_chem_xref=None*, *verbose=False*)
Convert padmet file to sbml file. Specificity: - ids are encoded for sbml using functions sbmlPlugin.convert_to_coded_id

Parameters

- **padmet** (*str or padmet.classes.PadmetSpec/PadmetRef*) – the pathname to the padmet file to convert, or PadmetSpec/PadmetRef object
- **output** (*str*) – the pathname to the sbml file to create
- **model_id** (*str or None*) – model id to set in sbml file
- **obj_fct** (*str*) – the identifier of the objection function, the reaction to test in FBA
- **sbml_lvl** (*int*) – the sbml level
- **sbml_version** (*int*) – the sbml version
- **verbose** (*bool*) – print informations

padmet.utils.connection.sbmlGenerator.**parse_mnx_chem_prop**(*mnx_chem_prop*)
#TODO

padmet.utils.connection.sbmlGenerator.**parse_mnx_chem_xref**(*mnx_chem_xref*)
#TODO

padmet.utils.connection.sbmlGenerator.**reaction_to_sbml**(*reactions*, *output*, *padmetRef*, *verbose=False*)
convert a list of reactions to sbml format based on a given padmet of reference. - ids are encoded for sbml using functions sbmlPlugin.convert_to_coded_id

Parameters

- **reactions** (*list*) – list of reactions ids
- **padmetRef** (*padmet.classes.PadmetRef*) – padmet of reference
- **output** (*str*) – the pathname to the sbml file to create

padmet.utils.connection.sbmlGenerator.**sbmlGenerator_cli**(*command_args*)

sbml_to_curation_form

Description: extract 1 reaction (if rxn_id) or a list of reactions (if rxn_file) from a sbml file to the form used in aureme for curation. For example use this script to extract specific missing reaction of a model to a just created metabolic network.

```
usage:
    padmet sbml_to_curation_form --sbml=FILE --output=FILE --rxn_id=ID [--
↪comment=STR] [--extract-gene] [-v]
    padmet sbml_to_curation_form --sbml=FILE --output=FILE --rxn_file=FILE [--
↪comment=STR] [--extract-gene] [-v]

options:
    -h --help     Show help.
    --sbml=FILE    path of the sbml.
    --output=FILE    form containing the reaction extracted, form used for manual␣
↪curation in aureme.
    --rxn_id=FILE    id of one reaction to extract
    --rxn_file=FILE    file of reactions ids to extract, 1 id by line.
    --extract-gene    If true, extract also genes associated to reactions.
    --comment=STR    comment associated to the reactions in the form. Used to track␣
↪sources of curation in aureme [default: "N.A"].
    -v   print info
```

padmet.utils.connection.sbml_to_curation_form.**command_help**()
Show help for analysis command.

padmet.utils.connection.sbml_to_curation_form.**sbml_to_curation**(*sbml_file*, *rxn_list*, *output*, *extract_gene=False*, *comment='N.A'*, *verbose=False*)
Read a sbml file, check if each reaction ids are in the sbml, if no, raise ValueError Then create the form. this form can then be used with manual_curation.py

Parameters

- **sbml_file** (*str*) – path to sbml file
- **rxn_list** (*list*) – list of reaction id, ids must be identic as in the sbml, carrefull to encoded ids.
- **output** (*str*) – path to the form to create
- **extract_gene** (*bool*) – if true extract genes association
- **comment** (*str*) – Comment why the reaction will be added in the network for traceability.
- **verbose** (*bool*) – if True print information

padmet.utils.connection.sbml_to_curation_form.**sbml_to_curation_form_cli**(*command_args*)

sbml_to_padmet

Description: There are 3 cases of convertion sbml to padmet:

1./ Creation of a reference database in padmet format from sbml(s) (or updating one with new(s) sbml(s)) First usage, padmetRef is the padmetRef to create or to update. If it's an update case, the output can be used to create a new padmet, if output None, will overwritte the input padmetRef.

2./ Creation of a padmet representing an organism in padmet format from sbml(s) (or updating one with new(s) sbml(s)) 2.A/ Without a database of reference: Second usage, padmetSpec is the padmetSpec to create or update. If it's an update case, the output can be used to create a new padmet, if output None, will overwritte the input padmetSpec.

2.B/ With a database of refence: Third usage, padmetSpec is the padmetSpec to create or update. If it's an update case, the output can be used to create a new padmet, if output None, will overwritte the input padmetSpec. padmetRef is the padmet representing the database of reference.

It is possible to define a specific policy and info for the padmet. To learn more about policy and info check doc of lib.padmetRef/Spec. if the ids of reactions/compounds are not the same between padmetRef and the sbml, it is possible to use a dictionnary of association (sbml_id padmetRef_id) with one line = 'id_sbml id_padmetRef' Finally if a reaction from sbml is not in padmetRef, it is possible to force the copy and creating a new reaction in padmetSpec with the arg -f

```
usage:
    padmet sbml_to_padmet --sbml=DIR/FILE --padmetRef=FILE [--output=FILE] [--db=STR]␣
↪[--version=STR] [-v]
    padmet sbml_to_padmet --sbml=DIR/FILE --padmetSpec=FILE [--output=FILE] [--source_
↪tool=STR] [--source_category=STR] [--source_id=STR] [-v]
    padmet sbml_to_padmet --sbml=DIR/FILE --padmetSpec=FILE  --padmetRef=FILE  [--
↪mapping=DIR/FILE] [--mapping_tag=STR] [--output=FILE] [--source_tool=STR] [--source_
↪category=STR] [--source_id=STR] [-v] [-f]

options:
    -h --help     Show help.
    --padmetSpec=FILE    path to the padmet file to update with the sbml. If there's␣
↪no padmetSpec, just specify the output
    --padmetRef=FILE    path to the padmet file representing to the database of␣
↪reference (ex: metacyc_18.5.padmet)
    --sbml=FILE    1 sbml file to convert into padmetSpec (ex: my_network.xml/sbml)␣
↪OR a directory with n SBML
    --output=FILE   pathanme to the new padmet file
    --mapping=FILE    dictionnary of association id_origin id_ref
    --mapping_tag=STR    if sbml is a folder, use a tag to define mapping files ex:␣
↪org1.sbml and org1_dict.csv, '_dict.csv' will be the mapping tag. [default: _dict.
↪csv]
    --db=STR    database name
    --version=STR    database version
    -v   print info
```

padmet.utils.connection.sbml_to_padmet.**command_help**()
Show help for analysis command.

padmet.utils.connection.sbml_to_padmet.**sbml_to_padmetRef**(*sbml*, *padmetRef_file*, *output=None*, *db='NA'*, *version='NA'*, *verbose=False*)

if padmetRef, not padmetSpec: if padmetRef exist, instance PadmetRef else init PadmetRef update padmetRef

if padmetSpec: if padmetRef, check if exist else raise Error if padmetSpec exist, instance PadmetSpec else init PadmetSpec update padmetSpec using padmetRef if padmetRef

#TODO

padmet.utils.connection.sbml_to_padmet.**sbml_to_padmetSpec**(*sbml*, *padmetSpec_file*, *padmetRef_file=None*, *output=None*, *mapping=None*, *mapping_tag='_dict.csv'*, *source_tool=None*, *source_category=None*, *db='NA'*, *version='NA'*, *verbose=False*)

Convert 1 - n sbml to padmet file. sbml var is file or dir padmetSpec_file: path to new padmet file to create or old padmet to update padmetRef_file: path to database of reference to use for data standardization output: path to new padmet file, if none, overwritte padmetSpec_file source_tool: tool used to create this sbml(s) ex Orthofinder source_category: Category of the tool ex: orthology if new padmet without padmetRef:

db: database used ex: metacyc, bigg version: version of the database, 23, 18...

if padmetRef, not padmetSpec: if padmetRef exist, instance PadmetRef else init PadmetRef update padmetRef

if padmetSpec: if padmetRef, check if exist else raise Error if padmetSpec exist, instance PadmetSpec else init PadmetSpec update padmetSpec using padmetRef if padmetRef

#TODO

padmet.utils.connection.sbml_to_padmet.**sbml_to_padmet_cli**(*command_args*)

sbml_to_sbml

Description: Create sbml from sbml. Use it to change sbml level.

```
usage:
    padmet sbml_to_sbml --input=FILE/FOLDER --output=FILE/FOLDER --new_sbml_lvl=STR [-
↪-cpu=INT]

option:
    -h --help    Show help.
    --input=FILE    path of the sbml file/folder to convert into sbml
    --output=FILE    path of the sbml file/folder to generate.
    --new_sbml_lvl=STR    level of the new sbml.
    --cpu=FILE    number of cpu used [default: 1].
    -v   print info.
```

padmet.utils.connection.sbml_to_sbml.**command_help**()

Show help for analysis command.

padmet.utils.connection.sbml_to_sbml.**from_sbml_to_sbml**(*input_sbml*, *output_sbml*, *new_sbml_level*, *cpu=1*)

Turn sbml to sbml.

Parameters

- **input_sbml** (*str*) – pathname to species sbml file/folder
- **output_sbml** (*str*) – pathname to output sbml file/folder
- **new_sbml_level** (*int*) – new sbml level
- **cpu** (*int*) – number of cpu

Returns pathname to output sbml file/folder

Return type str

padmet.utils.connection.sbml_to_sbml.**run_sbml_to_sbml**(*multiprocess_data*)
Turn sbml to sbml.

Parameters **multiprocess_data** (*dict*) – pathname to species sbml file, pathname to output sbml file, new sbml level

Returns True if sbml file exists

Return type bool

padmet.utils.connection.sbml_to_sbml.**sbml_to_padmet**(*sbml*, *db*, *version*, *source_tool*, *source_category*, *source_id*, *mapping*, *verbose*, *padmet_id*)
#TODO

padmet.utils.connection.sbml_to_sbml.**sbml_to_sbml_cli**(*command_args*)

wikiGenerator

Description: Contains all necessary functions to generate wikiPages from a padmet file and update a wiki online. Require WikiManager module (with wikiMate,Vendor)

```
usage:
    padmet wikiGenerator --padmet=FILE/DIR --output=DIR --wiki_id=STR [--
↪database=STR] [--padmetRef=FILE] [--log_file=FILE] [-v]
    padmet wikiGenerator --aureme_run=DIR --padmetSpec=ID -v

options:
    -h --help     Show help.
    --padmet=FILE    path to padmet file.
    --output=DIR    path to folder to create with all wikipages in subdir.
    --wiki_id=STR    id of the wiki.
    --padmetRef=FILE    path to padmet of reference, ex: metacyc_xx.padmet, if given,␣
↪able to calcul pathway rate completion.
    --log_file=FILE    log file from an aureme run, use this file to create a␣
↪wikipage with all the command used during the aureme run.
    --aureme_run=DIR    can use an aureme run as input, will use from config file␣
↪information for model_id and log_file and padmetRef.
    -v    print info.
```

padmet.utils.connection.wikiGenerator.**add_collapsible**(*text_array*, *title=None*)
#TODO

padmet.utils.connection.wikiGenerator.**add_property**(*properties*, *prop_id*, *prop_values*)
#TODO

padmet.utils.connection.wikiGenerator.**command_help**()
Show help for analysis command.

padmet.utils.connection.wikiGenerator.**copy_io_files**()

padmet.utils.connection.wikiGenerator.**createDirectory**(*output*, *verbose=False*)
create the folders genes, reactions, metabolites, pathways in the folder dirPath/ if already exist, it will replace old folders (and delete old files)

Parameters **output** (*str*) – path to output folder

padmet.utils.connection.wikiGenerator.**create_biological_page**(*category*, *page_id*, *page_dict_data*, *total_padmet_data*, *ext_link*, *output_file*, *padmetRef=None*, *verbose=False*)

#TODO

padmet.utils.connection.wikiGenerator.**create_log_page**(*log_file*, *output_folder*)

#TODO

padmet.utils.connection.wikiGenerator.**create_main**(*total_padmet_data*, *wiki_id*, *output_file*, *verbose=False*)

#TODO

padmet.utils.connection.wikiGenerator.**create_navigation_page**(*total_padmet_data*, *navigation_folder*, *verbose=False*)

#TODO

padmet.utils.connection.wikiGenerator.**create_venn**(*total_padmet_data*, *output_file*, *verbose=False*)

#TODO

padmet.utils.connection.wikiGenerator.**draw_ellipse**(*fig*, *ax*, *x*, *y*, *w*, *h*, *a*, *fillcolor*)

padmet.utils.connection.wikiGenerator.**draw_text**(*fig, ax, x, y, text, color=[0, 0, 0, 1]*)

padmet.utils.connection.wikiGenerator.**extract_padmet_data**(*padmetFile*, *total_padmet_data*, *global_pwy_rxn_dict=None*, *padmetRef=None*, *verbose=False*)

total_padmet_data: k in ['reaction', 'gene', 'organism', 'pathway', ...] if k = 'reaction', v = {'misc':{},'gene_assoc':}

For reaction in padmetFile:

if reaction_id not in total_padmet_data["reaction"].keys(): add total_padmet_data["reaction"][reaction_id][padmet_sou = dict()

else, add data only if differents from first

#TODO

padmet.utils.connection.wikiGenerator.**get_cmd_label**(*cmd*)

#TODO

padmet.utils.connection.wikiGenerator.**get_labels**(*data, fill=['number']*)

get a dict of labels for groups in data example: In [12]: get_labels([range(10), range(5,15), range(3,8)], fill=["number"]) Out[12]: {'001': '0', '010': '5', '011': '0', '100': '3', '101': '2', '110': '2', '111': '3'}

Parameters

- **data** (*list*) – data to get label for
- **fill** – ["number"|"logic"|"percent"]

Returns a dict of labels for different sets

Return type dict

padmet.utils.connection.wikiGenerator.**reduce_padmet_data**(*total_padmet_data*, *verbose=False*)

#TODO

padmet.utils.connection.wikiGenerator.**update_basic_attrib**(*node*, *current_node_dict*, *padmet_source*)

#TODO

padmet.utils.connection.wikiGenerator.**venn4**(*labels*, *names=['A', 'B', 'C', 'D']*, ***options*)

plots a 4-set Venn diagram

Parameters

- **labels** (*dict*) – a label dict where keys are identified via binary codes ('0001', '0010', '0100', ...), hence a valid set could look like: {'0001': 'text 1', '0010': 'text 2', '0100': 'text 3', ... }. unmentioned codes are considered as ''.
- **names** (*list*) – group names

Returns (Figure, AxesSubplot), pyplot Figure and AxesSubplot object

Return type set

padmet.utils.connection.wikiGenerator.**wikiGenerator**(*padmet*, *output*, *wiki_id*, *padmetRef=None*, *database=None*, *log_file=None*, *verbose=False*)

#TODO

padmet.utils.connection.wikiGenerator.**wikiGenerator_cli**(*command_args*)

padmet.utils.connection.wikiGenerator.**xrefLink**(*dataInArray*, *db*, *ids*)

#TODO

5.2.3 Exploration

Description:

#TODO

compare_padmet

Description: #Compare 1-n padmet and create a folder output with files: genes.tsv:

fieldnames = [gene, padmet_a, padmet_b, padmet_a_rxn_assoc, padmet_b_rxn_assoc] line = [gene-a, 1 (if in padmet_a), 1 (if in padmet_b), rxn-1;rxn-2 (names of reactions associated to gene-a in padmet_a), rxn-2]

reactions.tsv: fieldnames = [reaction, padmet_a, padmet_b, padmet_a_genes_assoc, padmet_b_genes_assoc, padmet_a_formula, padmet_b_formula] line = [rxn-1, 1 (if in padmet_a), 1 (if in padmet_b), 'gene-a;gene-b; gene-a, 'cpd-1 + cpd-2 => cpd-3', 'cpd-1 + cpd-2 => cpd-3']

pathways.tsv: fieldnames = [pathway, padmet_a_completion_rate, padmet_b_completion_rate, padmet_a_rxn_assoc, padmet_b_rxn_assoc] line = [pwy-a, 0.80, 0.30, rxn-a;rxn-b; rxn-a]

compounds.tsv: fieldnames = ['metabolite', padmet_a_rxn_consume, padmet_a_rxn_produce, padmet_b_rxn_consume, padmet_rxn_produce] line = [cpd-1, rxn-1,'',rxn-1,'']

```
usage:
    padmet compare_padmet --padmet=FILES/DIR --output=DIR [--padmetRef=FILE] [--cpu␣
↪INT] [-v]

option:
    -h --help    Show help.
    --padmet=FILES/DIR    pathname of the padmet files, sep all files by ',', ex: /
↪path/padmet1.padmet;/path/padmet2.padmet OR a folder
    --output=DIR    pathname of the output folder
    --padmetRef=FILE    pathanme of the database ref in padmet
    --cpu INT    number of CPU to use in multiprocessing
```

padmet.utils.exploration.compare_padmet.**command_help**()
Show help for analysis command.

padmet.utils.exploration.compare_padmet.**compare_padmet**(*padmet_path*, *output*, *padmetRef=None*, *verbose=False*, *number_cpu=None*)

#Compare 1-n padmet and create a folder output with files: genes.tsv:

fieldnames = [gene, padmet_a, padmet_b, padmet_a_rxn_assoc, padmet_b_rxn_assoc] line = [gene-a, 1 (if in padmet_a), 1 (if in padmet_b), rxn-1;rxn-2 (names of reactions associated to gene-a in padmet_a), rxn-2]

reactions.tsv: fieldnames = [reaction, padmet_a, padmet_b, padmet_a_genes_assoc, padmet_b_genes_assoc, padmet_a_formula, padmet_b_formula] line = [rxn-1, 1 (if in padmet_a), 1 (if in padmet_b), 'gene-a;gene-b; gene-a, 'cpd-1 + cpd-2 => cpd-3', 'cpd-1 + cpd-2 => cpd-3']

pathways.tsv: fieldnames = [pathway, padmet_a_completion_rate, padmet_b_completion_rate, padmet_a_rxn_assoc, padmet_b_rxn_assoc] line = [pwy-a, 0.80, 0.30, rxn-a;rxn-b; rxn-a]

compounds.tsv: fieldnames = ['metabolite', padmet_a_rxn_consume, padmet_a_rxn_produce, padmet_b_rxn_consume, padmet_rxn_produce] line = [cpd-1, rxn-1,'',rxn-1,'']

Parameters

- **padmet_path** (*str*) – pathname of the padmet files, sep all files by ',', ex: /path/padmet1.padmet;/path/padmet2.padmet OR a folder
- **output** (*str*) – pathname of the output folder
- **padmetRef** (*padmet.classes.PadmetRef*) – padmet containing the database of reference, need to calculat pathway completion rate
- **verbose** (*bool*) – if True print information

padmet.utils.exploration.compare_padmet.**compare_padmet_cli**(*command_args*)

padmet.utils.exploration.compare_padmet.**extract_information_padmet**(*file_path*, *padmetRef*, *verbose*)

padmet.utils.exploration.compare_padmet.**merge_dicts**(*element_dict*, *tmp_dict*)

compare_sbml

Description: compare reactions in 1-n or 2 sbml.

Returns if a reaction is missing

And if a reaction with the same id is using different species or different reversibility

```
usage:
    padmet compare_sbml --sbml=FILES/DIR --output=DIR

option:
    -h --help    Show help.
    --sbml FILES/DIR    pathname of the sbml files, sep all files by ',', ex: /path/
↪sbml1.sbml;/path/sbml2.sbml OR a folder
    --output DIR    pathname of the output folder
```

padmet.utils.exploration.compare_sbml.**command_help**()
Show help for analysis command.

padmet.utils.exploration.compare_sbml.**compare_multiple_sbml**(*sbml_path*, *output_folder*)
Compare 1-n sbml, create two output files reactions.tsv and metabolites.tsv with the reactions/metabolites in each sbml

Parameters

- **sbml_path** (*str*) – path to a folder containing sbmls or multiple sbml paths separated by a ','
- **output_folder** (*str*) – path to the output folder

padmet.utils.exploration.compare_sbml.**compare_rxn**(*rxn1*, *rxn2*)
compare two cobra reaction object and return (same_cpd, same_rev) same_cpd: bool, if true means same compounds consumed and produced same_reve: bool, if true means same direction of reaction (reversible or not)

Parameters

- **rxn1** (*cobra.model.reaction*) – reaction as cobra object
- **rxn2** (*cobra.model.reaction*) – reaction as cobra object

Returns (same_cpd (bool), same_rev (bool))

Return type tuple

padmet.utils.exploration.compare_sbml.**compare_sbml**(*sbml1_path*, *sbml2_path*)
Compare 2 sbml, print nb of metabolites and reactions. If reaction missing print reaction id, and reaction formula.

Parameters

- **sbml1_path** (*str*) – path to the first sbml file to compare
- **sbml2_path** (*str*) – path to the second sbml file to compare

padmet.utils.exploration.compare_sbml.**compare_sbml_cli**(*command_args*)

compare_sbml_padmet

Description: compare reactions in sbml and padmet file

```
usage:
    padmet compare_sbml_padmet --padmet=FILE --sbml=FILE

option:
```

(continues on next page)

(continued from previous page)

```
    -h --help    Show help.
    --padmet=FILE    path of the padmet file
    --sbml=FILE    path of the sbml file
```

padmet.utils.exploration.compare_sbml_padmet.**command_help**()
Show help for analysis command.

padmet.utils.exploration.compare_sbml_padmet.**compare_sbml_padmet**(*sbml_document*, *padmet*)
compare reactions ids in sbml vs padmet, return nb of reactions in both and reactions id not in sbml or not in padmet

Parameters

- **padmet** (*padmet.classes.PadmetSpec*) – padmet to udpate
- **sbml_file** (*libsbml.document*) – sbml document

padmet.utils.exploration.compare_sbml_padmet.**compare_sbml_padmet_cli**(*command_args*)

convert_sbml_db

Description: This tool is use the MetaNetX database to check or convert a sbml. Flat files from MetaNetx are required to run this tool. They can be found in the aureme workflow or from the MetaNetx website. To use the tool set:

mnx_folder= the path to a folder containing MetaNetx flat files. the files must be named as 'reac_xref.tsv' and 'chem_xref.tsv' or set manually the different path of the flat files with:

mnx_reac= path to the flat file for reactions

mnx_chem= path to the flat file for chemical compounds (species)

To check the database used in a sbml:

to check all element of sbml (reaction and species) set: to–map=all

to check only reaction of sbml set: to–map=reaction

to check only species of sbml set: to–map=species

To map a sbml and obtain a file of mapping ids to a given database set:

to-map: as previously explained

db_out: the name of the database target: ['metacyc', 'bigg', 'kegg'] only

output: the path to the output file

For a given sbml using a specific database.

Return a dictionnary of mapping.

the output is a file with line = reaction_id/or species in sbml, reaction_id/species in db_out database

ex: For a sbml based on kegg database, db_out=metacyc: the output file will contains for ex:

R02283 ACETYLORNTRANSAM-RXN

```
usage:
    padmet convert_sbml_db --mnx_reac=FILE --mnx_chem=FILE --sbml=FILE --to-map=STR [-
↪v]
    padmet convert_sbml_db --mnx_folder=DIR --sbml=FILE --to-map=STR [-v]
    padmet convert_sbml_db --mnx_folder=DIR --sbml=FILE --output=FILE --db_out=ID --
↪to-map=STR [-v]
```

(continues on next page)

(continued from previous page)

```
    padmet convert_sbml_db --mnx_reac=FILE --mnx_chem=FILE --sbml=FILE --output=FILE --
↪db_out=ID --to-map=STR [-v]

options:
    -h --help     Show help.
    --to-map=STR     select the part of the sbml to check or convert, must be in ['all
↪', 'reaction', 'species']
    --mnx_reac=FILE     path to the MetaNetX file for reactions
    --mnx_chem=FILE     path to the MetaNetX file for compounds
    --sbml=FILE     path to the sbml file to convert
    --output=FILE     path to the file containing the mapping, sep = "\t"
    --db_out=FILE     id of the output database in ["BIGG","METACYC","KEGG"]
    -v     verbose.
```

padmet.utils.exploration.convert_sbml_db.**check_sbml_db**(*sbml_file*, *to_map*, *verbose=False*, *mnx_reac_file=None*, *mnx_chem_file=None*, *mnx_folder=None*)

Check sbml database of a given sbml.

Parameters

- **sbml_file** (*str*) – path to the sbml file to convert
- **to_map** (*str*) – select the part of the sbml to check must be in ['all', 'reaction', 'species']
- **verbose** (*bool*) – if true: more info during process
- **mnx_reac_file** (*str*) – path to the flat file for reactions (can be None if given mnx_folder)
- **mnx_chem_file** (*str*) – path to the flat file for chemical compounds (species) (can be None if given mnx_folder)
- **mnx_folder** (*str*) – the path to a folder containing MetaNetx flat files

Returns (name of the best matching database, dict of matching)

Return type tuple

padmet.utils.exploration.convert_sbml_db.**command_help**()

Show help for analysis command.

padmet.utils.exploration.convert_sbml_db.**convert_sbml_db_cli**()

padmet.utils.exploration.convert_sbml_db.**get_from_mnx**(*mnx_dict*, *element_id*, *db_out*)

#TODO

padmet.utils.exploration.convert_sbml_db.**intern_mapping**(*id_to_map*, *db_out*, *_type*)

#TODO

padmet.utils.exploration.convert_sbml_db.**map_sbml**(*sbml_file*, *to_map*, *db_out*, *output*, *verbose=False*, *mnx_reac_file=None*, *mnx_chem_file=None*, *mnx_folder=None*)

map a sbml and obtain a file of mapping ids to a given database.

Parameters

- **sbml_file** (*str*) – path to the sbml file to convert
- **to_map** (*str*) – select the part of the sbml to check must be in ['all', 'reaction', 'species']
- **db_out** (*str*) – the name of the database target: ['metacyc', 'bigg', 'kegg'] only
- **output** (*str*) – path to the file containing the mapping, sep = " "
- **verbose** (*bool*) – if true: more info during process
- **mnx_reac_file** (*str*) – path to the flat file for reactions (can be None if given mnx_folder)
- **mnx_chem_file** (*str*) – path to the flat file for chemical compounds (species) (can be None if given mnx_folder)
- **mnx_folder** (*str*) – the path to a folder containing MetaNetx flat files

Returns (name of the best matching database, dict of matching)

Return type tuple

padmet.utils.exploration.convert_sbml_db.**mnx_reader**(*input_file*, *db_out*)
#TODO

dendrogram_reactions_distance

Description: Use reactions.tsv file from compare_padmet.py to create a dendrogram using a Jaccard distance.

From the matrix absence/presence of reactions in different species computes a Jaccard distance between these species. Apply a hierarchical clustering on these data with a complete linkage. Then create a dendrogram. Apply also intervene to create an upset graph on the data.

```
usage:
    padmet dendrogram_reactions_distance --reactions=FILE --output=FOLDER [--
↪padmetRef=STR] [--pvclust] [--upset=INT] [-v]

option:
    -h --help    Show help.
    --reactions=FILE    pathname of the file containing reactions in each species of␣
↪the comparison.
    --output=FOLDER    path to the output folder.
    --pvclust    launch pvclust dendrogram using R
    --padmetRef=STR    path to the padmet Ref file
    -u --upset=INT    number of cluster in the upset graph.
    -v    verbose mode.
```

padmet.utils.exploration.dendrogram_reactions_distance.**absent_and_specific_reactions**(*reaction*
out-
put_fola
out-
put_fola
out-
put_fola
or-
gan-
isms)

Compare all cluster one against another.

Parameters

- **reactions_dataframe** (*pandas.DataFrame*) – dataframe containing absence/presence of reactions in organism
- **output_folder_tree_cluster** (*str*) – path to output tree cluster folder
- **output_folder_specific** (*str*) – path to output folder with specific reactions for each species
- **output_folder_absent** (*str*) – path to output folder with absent reactions for each species
- **organisms** (*list*) – organisms names

padmet.utils.exploration.dendrogram_reactions_distance.**add_dendrogram_node_label**(*reaction_dend node_list, reactions_clust, len_longest_c*

Using cluster nodes, add label and reactions number on each node of the dendrogram. This function comes from this answer on stackoverflow: https://stackoverflow.com/a/43519473

Parameters

- **reactions_dataframe** (*pandas.DataFrame*) – dataframe containing absence/presence of reactions in organism
- **node_list** (*list*) – cluster nodes
- **reactions_clust** (*dictionary*) – reactions in each cluster of the tree
- **len_longest_cluster_id** (*int*) – reactions in each cluster of the tree

padmet.utils.exploration.dendrogram_reactions_distance.**command_help**()

Show help for analysis command.

padmet.utils.exploration.dendrogram_reactions_distance.**comparison_cluster**(*reactions_clust, output_folder_comparison*)

Compare all cluster one against another.

Parameters

- **reactions_clust** (*dictionary*) – reactions in each cluster of the tree
- **output_folder_comparison** (*str*) – path to output folder

padmet.utils.exploration.dendrogram_reactions_distance.**create_cluster**(*reactions_dataframe, absence_presence_matrix, linkage_matrix*)

Cut the dendrogram to create clusters.

Parameters

- **reactions_dataframe** (*pandas.DataFrame*) – dataframe containing absence/presence of reactions in organism
- **absence_presence_matrix** (*pandas.DataFrame*) – transposition of the reactions dataframe
- **linkage_matrix** (*ndarray*) – linkage matrix

Returns **dendrogram_fclusters** – {number used to split the linkage matrix: ndarray with the corresponding clusters}

Return type dictionary

padmet.utils.exploration.dendrogram_reactions_distance.**create_intersection_files**(*root*, *cluster_leaf_speci*
reactions_datafra
output_folder_tre
metacyc_to_ecs)

Create intersection files.

Parameters

- **root** (*root*) – root of the xml tree
- **cluster_leaf_species** (*dictionary*) – for each leaf give the organisms in it
- **reactions_dataframe** (*pandas.DataFrame*) – dataframe containing absence/presence of reactions in organism
- **output_folder_tree_cluster** (*str*) – path to the output folder
- **metacyc_to_ecs** (*dictionary*) – mapping of metayc reaction to EC number

Returns **reactions_clust** – reactions in each cluster of the tree

Return type dictionary

padmet.utils.exploration.dendrogram_reactions_distance.**create_intervene_graph**(*absence_presence_*
reactions_dataframe, *temp_data_folder*, *path_to_intervene*
output_folder_upset, *dendrogram_fclusters*, *k*, *verbose=False*)

Create an upset graph. Deprecated function, no we use supervenn look at create_supervenn function.

Parameters

- **absence_presence_matrix** (*pandas.DataFrame*) – transposition of the reactions dataframe
- **reactions_dataframe** (*pandas.DataFrame*) – dataframe containing absence/presence of reactions in organism
- **temp_data_folder** (*str*) – temporary data folder
- **path_to_intervene** (*str*) – path to intervene bin
- **output_folder_upset** (*str*) – path to output folder

- **dendrogram_fclusters** (*dictionary*) – {number used to split the linkage matrix: ndarray with the corresponding clusters}
- **k** (*int*) – number of cluster to create

padmet.utils.exploration.dendrogram_reactions_distance.**create_pvclust_dendrogram**(*reaction_file*, *output_folder*)

padmet.utils.exploration.dendrogram_reactions_distance.**create_supervenn**(*absence_presence_matrix*, *reactions_dataframe*, *output_folder_upset*, *dendrogram_fclusters*, *k*, *verbose=False*)

Create an supervenn graph.

Parameters

- **absence_presence_matrix** (*pandas.DataFrame*) – transposition of the reactions dataframe
- **reactions_dataframe** (*pandas.DataFrame*) – dataframe containing absence/presence of reactions in organism
- **output_folder_upset** (*str*) – path to output folder
- **dendrogram_fclusters** (*dictionary*) – {number used to split the linkage matrix: ndarray with the corresponding clusters}
- **k** (*int*) – number of cluster to create

padmet.utils.exploration.dendrogram_reactions_distance.**dendrogram_reactions_distance_cli**(*co*

padmet.utils.exploration.dendrogram_reactions_distance.**getNewick**(*node*, *newick*, *parentdist*, *leaf_names*)

Create a newick file from the root node of the dendrogram. This function comes from this answer on stackoverflow: https://stackoverflow.com/a/31878514.

Parameters

- **node** (*scipy.cluster.hierarchy.ClusterNode*) – root ClusterNode of the scipy tree
- **newick** (*str*) – newick string
- **parentdist** (*str*) – root ClusterNode distance from the linkage matrix
- **leaf_names** (*list*) – list of organism names

padmet.utils.exploration.dendrogram_reactions_distance.**hclust_to_xml**(*linkage_matrix*)

Using a distance matrix from scipy linkage, create a xml tree corresponding to the hierarchical clustering. Return the root of the tree.

Parameters **linkage_matrix** (*ndarray*) – linkage matrix

Returns root of the xml tree

Return type root

padmet.utils.exploration.dendrogram_reactions_distance.**pvclust_dendrogram**(*reactions_dataframe*, *organisms*, *output_folder*)

Using a distance matrix, pvclust R package (with rpy2 package) create a dendrogram with bootstrap values.

Parameters

- **reactions_dataframe** (*pandas.DataFrame*) – Reactions absence/presence matrix
- **organisms** (*list*) – organisms names
- **output_folder** (*str*) – path to the output folder

padmet.utils.exploration.dendrogram_reactions_distance.**reaction_figure_creation**(*reaction_file*, *output_folder*, *upset_cluster=No*, *padmetRef_file=None*, *pvclust=None*, *verbose=False*)

Create dendrogram, upset figure (if upset argument) and compare reactiosn in species.

Parameters

- **reaction_file** (*str*) – path to reaction file
- **upset_cluster** (*int*) – the number of cluster you want in the intervene figure
- **output_folder** (*str*) – path to output folder
- **padmet_ref_file** (*str*) – path to padmet ref file
- **pvclust** (*bool*) – boolean to launch or not R pvclust dendrogram

flux_analysis

Description: 1./ Run flux balance analyse with cobra package on an already defined reaction. Need to set in the sbml the value 'objective_coefficient' to 1. If the reaction is reachable by flux: return the flux value and the flux value for each reactant of the reaction. If not: only return the flux value for each reactant of the reaction. If a reactant has a flux of '0' this means that it is not reachable by flux (and maybe topologically). To unblock the reaction it is required to fix the metabolic network by adding/removing reactions until all reactant are reachable.

2./If seeds and targets given as sbml files with only compounds. Will also try to use the Menetools library to make a topologicall analysis. Topological reachabylity of the targets compounds from the seeds compounds.

3./ If –all_species: will test flux reachability of all the compounds in the metabolic network (may take several minutes)

```
usage:
    padmet flux_analysis --sbml=FILE
    padmet flux_analysis --sbml=FILE --seeds=FILE --targets=FILE [--all_species]
    padmet flux_analysis --sbml=FILE --all_species

option:
    -h --help    Show help.
    --sbml=FILE    pathname to the sbml file to test for fba and fva.
    --seeds=FILE    pathname to the sbml file containing the seeds (medium).
    --targets=FILE    pathname to the sbml file containing the targets.
    --all_species    allow to make FBA on all the metabolites of the given model.
```

padmet.utils.exploration.flux_analysis.**command_help**()
Show help for analysis command.

padmet.utils.exploration.flux_analysis.**fba_on_targets**(*allspecies*, *model*)
for each specie in allspecies, create an objective function with the current species as only product and try to optimze the model and get flux.

Parameters

- **allSpecies** (*list*) – list of species ids to test
- **model** (*cobra.model*) – Cobra model from a sbml file

padmet.utils.exploration.flux_analysis.**flux_analysis**(*sbml_file*, *seeds_file=None*, *targets_file=None*, *all_species=False*)
1./ Run flux balance analyse with cobra package on an already defined reaction. Need to set in the sbml the value 'objective_coefficient' to 1. If the reaction is reachable by flux: return the flux value and the flux value for each reactant of the reaction. If not: only return the flux value for each reactant of the reaction. If a reactant has a flux of '0' this means that it is not reachable by flux (and maybe topologically). To unblock the reaction it is required to fix the metabolic network by adding/removing reactions until all reactant are reachable.

2./If seeds and targets given as sbml files with only compounds. Will also try to use the Menetools library to make a topologicall analysis. Topological reachabylity of the targets compounds from the seeds compounds.

3./ If –all_species: will test flux reachability of all the compounds in the metabolic network (may take several minutes)

Parameters

- **sbml_file** (*str*) – path to sbml file to analyse
- **seeds_file** (*str*) – path to sbml file with only compounds representing the seeds/growth medium
- **targets_file** (*str*) – path to sbml file with only compounds representing the targets to reach
- **all_species** (*bool*) – if True will try to create obj function for each compound and return which are reachable by flux.

padmet.utils.exploration.flux_analysis.**flux_analysis_cli**(*command_args*)

get_pwy_from_rxn

Description: From a file containing a list of reaction, return the pathways where these reactions are involved. ex: if rxn-a in pwy-x => return, pwy-x; all rxn ids in pwy-x; all rxn ids in pwy-x FROM the list; ratio

```
usage:
    padmet get_pwy_from_rxn --reaction_file=FILE --padmetRef=FILE  --output=FILE

options:
    -h --help     Show help.
    --reaction_file=FILE    pathname of the file containing the reactions id, 1/line
    --padmetRef=FILE    pathname of the padmet representing the database.
    --output=FILE    pathname of the file with line = pathway id, all reactions id,␣
↪reactions ids from reaction file, ratio. sep = "\t"
```

padmet.utils.exploration.get_pwy_from_rxn.**command_help**()
Show help for analysis command.

padmet.utils.exploration.get_pwy_from_rxn.**dict_pwys_to_file**(*dict_pwy*, *output*)
Create csv file from dict_pwy. dict_pwy is obtained with extract_pwys()

Parameters

- **dict_pwy** (*dict*) – dict, k=pathway_id, v=dict: k in [total_rxn, rxn_from_list, ratio ex: {pwy-x:{'total_rxn':[a,b,c], rxn_from_list:[a], ratio:1/3}}
- **output** (*str*) – path to output file

padmet.utils.exploration.get_pwy_from_rxn.**extract_pwys**(*padmet*, *reactions*)
#extract from padmet pathways containing 1-n reactions from a set of reactions 'reactions' Return a dict of data. dict, k=pathway_id, v=dict: k in [total_rxn, rxn_from_list, ratio ex: {pwy-x:{'total_rxn':[a,b,c], rxn_from_list:[a], ratio:1/3}}

Parameters

- **padmet** (*padmet.classes.PadmetSpec*) – padmet to udpate
- **reactions** (*set*) – set of reactions to match with pathways

Returns dict, k=pathway_id, v=dict: k in [total_rxn, rxn_from_list, ratio ex: {pwy-x:{'total_rxn':[a,b,c], rxn_from_list:[a], ratio:1/3}}

Return type dict

padmet.utils.exploration.get_pwy_from_rxn.**get_pwy_from_rxn**(*padmet*, *reaction_file*, *output*)

padmet.utils.exploration.get_pwy_from_rxn.**get_pwy_from_rxn_cli**(*command_args*)

padmet_stats

Description: From a file containing a list of reaction, return the pathways where these reactions are involved. ex: if rxn-a in pwy-x => return, pwy-x; all rxn ids in pwy-x; all rxn ids in pwy-x FROM the list; ratio

```
usage:
    padmet get_pwy_from_rxn --reaction_file=FILE --padmetRef=FILE  --output=FILE

options:
    -h --help     Show help.
    --reaction_file=FILE    pathname of the file containing the reactions id, 1/line
    --padmetRef=FILE    pathname of the padmet representing the database.
    --output=FILE    pathname of the file with line = pathway id, all reactions id,␣
↪reactions ids from reaction file, ratio. sep = "\t"
```

padmet.utils.exploration.get_pwy_from_rxn.**command_help**()
Show help for analysis command.

padmet.utils.exploration.get_pwy_from_rxn.**dict_pwys_to_file**(*dict_pwy*, *output*)
Create csv file from dict_pwy. dict_pwy is obtained with extract_pwys()

Parameters

- **dict_pwy** (*dict*) – dict, k=pathway_id, v=dict: k in [total_rxn, rxn_from_list, ratio ex: {pwy-x:{'total_rxn':[a,b,c], rxn_from_list:[a], ratio:1/3}}
- **output** (*str*) – path to output file

padmet.utils.exploration.get_pwy_from_rxn.**extract_pwys**(*padmet*, *reactions*)
#extract from padmet pathways containing 1-n reactions from a set of reactions 'reactions' Return a dict of data. dict, k=pathway_id, v=dict: k in [total_rxn, rxn_from_list, ratio ex: {pwy-x:{'total_rxn':[a,b,c], rxn_from_list:[a], ratio:1/3}}

Parameters

- **padmet** (*padmet.classes.PadmetSpec*) – padmet to udpate
- **reactions** (*set*) – set of reactions to match with pathways

Returns dict, k=pathway_id, v=dict: k in [total_rxn, rxn_from_list, ratio ex: {pwy-x:{'total_rxn':[a,b,c], rxn_from_list:[a], ratio:1/3}}

Return type dict

padmet.utils.exploration.get_pwy_from_rxn.**get_pwy_from_rxn**(*padmet*, *reaction_file*, *output*)

padmet.utils.exploration.get_pwy_from_rxn.**get_pwy_from_rxn_cli**(*command_args*)

padmet_stats

Description: Create a padmet stats file containing the number of pathways, reactions, genes and compounds inside the padmet.

The input is a padmet file or a folder containing multiple padmets.

Create a tsv file named padmet_stats.tsv where the script have been launched.

usage: padmet padmet_stats –padmet=FILE –output=FOLDER

option: -h –help Show help. -p –padmet=FILE padmet file or folder containing padmet(s). -o –output=FOLDER path to output folder.

padmet.utils.exploration.padmet_stats.**command_help**()
Show help for analysis command.

padmet.utils.exploration.padmet_stats.**compute_stats**(*padmet_file_folder*, *output_folder*)
Count reactions/pathways/compounds/genes in padmet(s).

Parameters

- **padmet_file_folder** (*str*) – path to the padmet file/folder to analyze
- **output_folder** (*str*) – path to the output folder

padmet.utils.exploration.padmet_stats.**orthology_result**(*padmet_file*, *padmet_names*)
Count reactions/pathways/compounds/genes in a padmet file.

Parameters

- **padmet_file** (*str*) – path to a padmet file

- **padmet_names** (*list*) – all the padmet filenames

Returns Number of reactions given by the other species

Return type dictionary

padmet.utils.exploration.padmet_stats.**padmet_stat**(*padmet_file*)
Count reactions/pathways/compounds/genes in a padmet file.

Parameters **padmet_file** (*str*) – path to a padmet file

Returns [path to padmet, number of pathways, number of reactions, number of genes, number of compounds]

Return type list

padmet.utils.exploration.padmet_stats.**padmet_stats_cli**(*command_args*)

prot2genome

Description: Prot2Genome contains functions used for blast analysis and padmet enrichment

usage: padmet prot2genome –query_faa=FILE –query_ids=FILE/STR –subject_gbk=FILE –subject_fna=FILE –subject_faa=FILE –output_folder=FILE [–cpu=INT] [blastp] [tblastn] [debug] padmet prot2genome –query_faa=FILE –query_ids=FILE/STR –subject_gbk=FILE –subject_fna=FILE –subject_faa=FILE –output_folder=FILE –exonerate=PATH [–cpu=INT] [blastp] [tblastn] [debug] padmet prot2genome –padmet=FOLDER –output=FOLDER padmet prot2genome –studied_organisms=FOLDER –output=FOLDER padmet prot2genome –run=FOLDER –padmetRef=FILE [–cpu=INT] [debug]

From aucome run fromAucome(): -1. Extract specifique reactions in spec_reactions folder with extractReactions() -2. Extract genes from spec_reactions files with extractGenes() -3. Run tblastn + exonerate with runAllAnalysis()

options: –query_faa=FILE #TODO. –query_ids=FILE/STR #TODO. –subject_gbk=FILE #TODO. –subject_fna=FILE #TODO. –subject_faa=FILE #TODO. –output_folder=FILE #TODO. –cpu=INT Number of cpu to use for the multiprocessing (if none use 1 cpu). [default: 1] blastp #TODO. tblastn #TODO. debug #TODO.

padmet.utils.exploration.prot2genome.**analysisOutput**(*analysis_result*, *analysis_output*)

padmet.utils.exploration.prot2genome.**cleanTmp**(*tmp_folder*)
Remove all files from tmp folder

Parameters **tmp_folder** (*str*) – path to tmp folder where to create faa of each gene to analyse

padmet.utils.exploration.prot2genome.**command_help**()
Show help for analysis command.

padmet.utils.exploration.prot2genome.**createPadmet**(*dict_args*)
function used in mp_createPadmet by each worker the Pool padmet are updated using funciton add_delete_rxn from padmet.utils.connection.manual_curation

padmet.utils.exploration.prot2genome.**createSeqFromTblastn**(*subject_fna*, *sseq_seq_faa*, *exonerate_target_id*, *start_match*, *end_match*)
Use the result from the tBlastn to extract a region from the subject genome. The region extracted corresponds to the match region and 10kb before and 10kb after.

Parameters

- **subject_fna** (*str*) – path to subject fasta sequence (genome)
- **sseq_seq_faa** (*str*) – path to output fasta sequence
- **exonerate_target_id** (*str*) – ID of the contig/scaffold/chromosome where a match has been found
- **start_match** (*int*) – start of the match
- **end_match** (*int*) – end of the match

padmet.utils.exploration.prot2genome.**extractAnalysis**(*blast_analysis_folder*, *spec_reactions_folder*, *output_folder*)

For each analysis output in blast analysis folder, obtained with runAllAnalysis() 1./ Extract orthologues hit 2./ For each specific reactions from spec_reactions_folder, if all genes of a reactions got ortho hit

add reaction to reactions_to_add

Parameters

- **blast_analysis_folder** (*str*) – path folder with all blast analysis output files
- **spec_reactions_folder** (*str*) – path folder with all files containing specific reactions
- **output_folder** (*str*) – path folder where to extract all reactions to add

padmet.utils.exploration.prot2genome.**extractGenes**(*reactions_file*)

Extract genes ids and return a list from reactions_file obtained with extractReactions()

Parameters **reactions_file** (*str*) – path to reaction file

padmet.utils.exploration.prot2genome.**extractReactions**(*dict_args*)

function used in mp_cextractReactions by each worker the Pool for org_a.padmet and org_b.padmet:

1./ extract reactions and specific reactiosn (not in a, not in b) 2./ extract genes associated to specific reactions 3./ Select only reactions if they are from annotation rxn-1 in org_a but not in org_b, if rxn-1 doesn't come from org_a annotation, skip the reaction 4./ create output file: header = ["reaction_id", "genes_ids", "sources"]

padmet.utils.exploration.prot2genome.**extract_sequence**(*exonerate_output*, *exonerate_sequence*)

Extract protein sequence from exonerate ouput.

padmet.utils.exploration.prot2genome.**fromAucome**(*run_folder*, *cpu*, *padmetRef*, *blastp=True*, *tblastn=True*, *exonerate=True*, *keep_tmp=False*, *debug=False*)

This function fit an AuCoMe run. Select a aucome run folder and then the function will: 1./ For each couple of studied organisms, extract specific reactions

ex: For org A and org B, extract reactions in org A but not in org B and vice versa

2./ Then for each specific reactions, extract genes associated and run blastp, tblastn and exonerate 3./ For each reaction, for all genes associated, if no blastp match but tblastn and exonerate hit select the reaction as a hit 4./ Create a new padmet file with the new reactions to add within

Parameters

- **run_folder** (*str*) – path to aucome run folder
- **cpu** (*int*) – number of cpu to use for multiprocessing steps

- **padmetRef** (*str*) – path to padmetRef from where to extract and add the new reactions to create new padmet files
- **blastp** (*bool*) – If true run blastp during analysis
- **tblastn** (*bool*) – If true run tblastn during analysis
- **exonerate** (*bool*) – If true run exonerate during analysis, tblastn must also be True
- **keep_tmp** (*bool*) – If true keep temporary files of analysis (with predicted gene sequence)
- **debug** (*bool*) – if true, print all raw informations of analysis

padmet.utils.exploration.prot2genome.**mp_createPadmet**(*reactions_to_add_folder*, *padmet_folder*, *output_folder*, *padmetRef*, *pool*, *verbose=False*)

Update all padmet in padmet_folder with reactions to add from file in reactiosn_to_add_folder, the informations of the reactions are extracted from padmetRef as unique source ex: for padmet_folder/org_a.padmet, select reactions_to_add_folder/org_a.tsv, add each reactions listed in this file based on padmetRef to create output_folder/org_a.padmet Create the padmet files in multiprocess, the more cpu the more new padmet files will be created faster

Parameters

- **reactions_to_add_folder** (*str*) – path folder with all files containing reactions to add for each studied organism
- **padmet_folder** (*str*) – path to folder with all padmet files of studied organism
- **output_folder** (*str*) – path to output folder where to create new padmet files
- **padmetRef** (*str*) – path to padmetRef from where to extract and add the new reactions to create new padmet files
- **pool** (*Pool object*) – pool object of multiprocessing
- **verbose** (*bool*) – verbose

padmet.utils.exploration.prot2genome.**mp_extractReactions**(*padmet_folder*, *output_folder*, *pool*)

From a folder of padmet files, create all dual combination and extract specific reactions to create a file in output_folder ex: in padmet_folder: org_a.padmet, org_b.padmet, create: output_folder: org_a_vs_org_b.tsv and org_b_vs_org_a.tsv

Parameters

- **padmet_folder** (*str*) – path to folder with all padmet files of studied organism
- **output_folder** (*str*) – path to output folder where to extract specific reactions
- **pool** (*Pool object*) – pool object of multiprocessing

padmet.utils.exploration.prot2genome.**mp_runAnalysis**(*spec_reactions_folder*, *studied_organisms_folder*, *output_folder*, *tmp_folder*, *pool*, *blastp*, *tblastn*, *exonerate*, *keep_tmp*, *debug*)

Run different blast analysis based on files representing specific reactions of 2 padmet files. For each specific reaction file in spec_reactions_folder (ex: org_a_vs_org_b.tsv):

1./ search for: faa file of org_a (studied_organisms_folder/org_a/org_a.faa) gbk file of org_b (studied_organisms_folder/org_b/org_b.gbk) faa file of org_b (studied_organisms_folder/org_b/org_b.faa) fna file of org_b (studied_organisms_folder/org_b/org_b.fna)

if fna doesn't exist create it

2./ if output file (blast_analysis_folder/org_a_VS_org_b.tsv) doesn't already exist run analysis 3./ extracts all genes ids from specific reaction file with fct extractGenes() 4./ Run blastp, tblastn, exonerate on gene_id.faa vs target.faa / fna with runAllAnalysis() 5./ Create analysis output The analysis create a lot of temp files, all are in tmp_folder wich is cleaned after all loop

Parameters

- **spec_reactions_older** (*str*) – path folder with all files containing specific reactions
- **studied_organisms_folder** (*str*) – path to folder with all data of studied organisms. Folder contains 1 folder by org with name as org name, in each: org.gbk,org.faa,org.fna
- **output_folder** (*str*) – path to output folder where to extract blast analysis
- **tmp_folder** (*str*) – path to tmp folder where to create faa of each gene to analyse
- **pool** (*Pool object*) – pool object of multiprocessing
- **blastp** (*bool*) – If true run blastp during analysis
- **tblastn** (*bool*) – If true run tblastn during analysis
- **exonerate** (*bool*) – If true run exonerate during analysis, tblastn must also be True
- **keep_tmp** (*bool*) – If true keep temporary files of analysis (with predicted gene sequence)
- **debug** (*bool*) – if true, print all raw informations of analysis

padmet.utils.exploration.prot2genome.**prot2genome_cli**(*command_args*)

padmet.utils.exploration.prot2genome.**runAllAnalysis**(*dict_args*)

For a given gene query id:

1/ extract from query_faa the sequence and create a faa file output_folder/query_id.faa If isoforms found, also search for each specific isoform

2/ if blastp, run blastp; if tblastn, run tblastn; if exonerate and tblastn has hit, run exonerate Run all of them and extract output as dict of data

Returns list of dict with all analysis output

Return type list

padmet.utils.exploration.prot2genome.**runBlastp**(*query_seq_faa, subject_faa, header=['sseqid', 'evalue', 'bitscore'], debug=False*)

Run blastp on querry_seq vs subectj faa and return output based on header Use NcbiblastpCommandline fct and extract output Extract 1st best hit based on bitscore

Parameters

- **query_seq_faa** (*str*) – path to query fasta sequence
- **subject_faa** (*str*) – path to subject fasta sequence
- **header** (*list*) – output format of blastp
- **debug** (*bool*) – if true print all raw blastp output

Returns dict of the best blastp hit, add '**blastp_**' tag, or empty dict if no hit

Return type dict

padmet.utils.exploration.prot2genome.**runExonerate**(*query_seq_faa*, *sseq_seq_faa*, *output*, *debug=False*)

Run exonerate on querry_seq vs subject faa Exonerate must be installed, and the global var PATH must be update with the exonerate/bin/ command 'exonerate' should work from shell sseq_seq_faa is obtained after tblastn run based on tblastn_sseqid value

Parameters

- **query_seq_faa** (*str*) – path to query fasta sequence
- **sseq_seq_faa** (*str*) – path to subject faa sequence
- **output** (*str*) – path to exonerate output
- **debug** (*bool*) – if true print all raw exonerate output

Returns dict of the best exonerate hit, add '**exonerate_**' tag, or empty dict if no hit

Return type dict

padmet.utils.exploration.prot2genome.**runSearchOnProteome**(*proteome_orgA*, *genome_orgB*, *output_folder*, *proteome_orgB=None*)

From a proteome of OrgA search for missing structural annotation in genome of OrgB. First launch Blastp between proteome of OrgA and proteome of OrgB. Then launch tBlastn between proteome of OrgA and genome of OrgB to find matches. Use the best match to extract a region from the genome of OrgB. Then launch Exonerate on this region using the sequence of OrgA.

Parameters

- **proteome_orgA** (*str*) – path to fasta file of proteome of OrgA
- **genome_orgB** (*str*) – path to fasta file of genome of OrgB
- **output_folder** (*str*) – path to output folder
- **proteome_orgB** (*str*) – path to fasta file of proteome of OrgB

padmet.utils.exploration.prot2genome.**runTblastn**(*query_seq_faa*, *subject_fna*, *header=['sseqid', 'evalue', 'bitscore', 'sstart', 'send']*, *debug=False*)

Run tblastn on querry_seq vs subectj fna and return output based on header Use NcbitblastnCommandline fct and extract output Extract 1st best hit based on bitscore

Parameters

- **query_seq_faa** (*str*) – path to query fasta sequence
- **subject_fna** (*str*) – path to subject fna sequence
- **header** (*list*) – output format of tblastn
- **debug** (*bool*) – if true print all raw tblastn output

Returns dict of the best tblastn hit, add '**tblastn_**' tag, or empty dict if no hit

Return type dict

visu_path

Description: Allows to visualize a pathway in padmet network.

Color code: reactions associated to the pathway, present in the network: lightgreen reactions associated to the pathway, not present in the network: red compounds: skyblue

```
usage:
    padmet visu_path --padmetSpec=FILE/FOLDER --padmetRef=FILE --pathway=ID --
↪output=FILE [--hide-currency] [--level=STR]

options:
    -h --help     Show help.
    --padmetSpec=FILE/FOLDER    pathname to the PADMet file of the network or to a␣
↪folder containing multiple padmets.
    --padmetRef=FILE    pathname to the PADMet file of the db of reference.
    --pathway=ID    pathway id to visualize, can be multiple pathways separated by a
↪",".
    --output=FILE    pathname to the output file (extension can be .png or .svg).
    --hide-currency    hide currency metabolites.
    --level=STR    level of precision for the visualization (compound or pathway). By␣
↪default visualization uses "compound".
```

padmet.utils.exploration.visu_path.**command_help**()
Show help for analysis command.

padmet.utils.exploration.visu_path.**visu_path_cli**(*command_args*)

padmet.utils.exploration.visu_path.**visu_path_compounds**(*padmet_pathname*, *padmet_ref_pathname*, *pathway_ids*, *output_file*, *hide_currency_metabolites=None*)
Extract reactions from pathway and create a comppound/reaction graph.

Parameters

- **padmet_pathname** (*str*) – pathname of the padmet file or a folder containing multiple padmet
- **padmet_ref_pathname** (*str*) – pathname of the padmetRef file
- **pathway_ids** (*str*) – name of the pathway (can be multiple pathways separated by a ‘,’)
- **output_file** (*str*) – pathname of the output picture (extension can be .png or .svg)
- **hide_currency_metabolites** (*bool*) – hide currency metabolites

padmet.utils.exploration.visu_path.**visu_path_pathways**(*padmet_pathname*, *padmet_ref_pathname*, *pathway_ids*, *output_file*)
Extract reactions from pathway and create a comppound/reaction graph.

Parameters

- **padmet_pathname** (*str*) – pathname of the padmet file or a folder containing multiple padmet
- **padmet_ref_pathname** (*str*) – pathname of the padmetRef file
- **pathway_ids** (*str*) – name of the pathway (can be multiple pathways separated by a ‘,’)
- **output_file** (*str*) – pathname of the output picture (extension can be .png or .svg)
- **hide_compounds** (*bool*) – hide common compounds (like water or proton)

5.2.4 Management

Description:

#TODO

manual_curation

Description: Update a padmetSpec by filling specific forms.

1./ Create new reaction(s) to padmet file.

- Get the template form with –template_new_rxn
- Fill the template
- set –data as path to the filled template

2./ Add reaction(s) from padmetRef or remove reactions(s).

- Get the template form with –template_add_delete_rxn
- Fill the template
- set –date as path to the filled template

Update padmetSpec and create a new padmet (new_padmet) or overwrite the input

```
usage:
    padmet manual_curation --padmetSpec=FILE --data=FILE [--padmetRef=FILE] [--
↪output=FILE] [--tool=STR] [--category=STR] [-v]
    padmet manual_curation --template_new_rxn=FILE
    padmet manual_curation --template_add_delete_rxn=FILE

option:
    -h --help    Show help.
    --padmetSpec=FILE    path to the padmet to update
    --padmetRef=FILE    path of the padmet representing the reference database
    --data=FILE    path to the form with data for curation
    --output=FILE    path to the output. if None. Overwriting padmetSpec
    --tool=STR    specification of the tool used to allow this curation: ex a tool of␣
↪gapfilling (meneco)
    --category=STR    specification of the category of curation: ex if a reaction is␣
↪added based on annotation info, use 'annotation'
    --template_new_rxn=FILE    create a form used to create new reaction, use this␣
↪form as input for 'data' option
    --template_add_delete_rxn=FILE    create a form used to add or delete reaction,␣
↪use this form as input for 'data' option
    -v    print info
```

padmet.utils.management.manual_curation.**add_delete_rxn**(*data_file*, *padmetSpec*, *output*, *padmetRef=None*, *source=None*, *tool=None*, *category='MANUAL'*, *verbose=False*)

Read a data_file (form created with template_add_delete and filed), for each reaction if column 'Action' == 'add':

add the reaction from padmetRef to padmetSpec.

elif column 'Action' == 'delete': remove the reaction

Can't add a reaction without a padmetRef !

the source ensure the traceability of the reaction, its a simple tag ex 'pathway_XX_update' if not given the filename of data_file will be used. if a tool was used to infer the reaction, define tool='name_of_the_tool'

Parameters

- **data_file** (*str*) – path to file based on template_new_rxn()
- **padmetSpec** (*padmet.classes.PadmetSpec*) – padmet to update
- **padmetRef** (*padmet.classes.PadmetRef*) – padmet containing the database of reference
- **output** (*str*) – path to the new padmet file
- **source** (*str*) – tag associated to the new reactions to create and add, used for traceability
- **tool** (*str*) – The eventual tool used to infer the reactions to create and add
- **category** (*str*) – The default category of the reaction added manually is 'MANUAL'. Must not be changed.
- **verbose** (*bool*) – if True print information

padmet.utils.management.manual_curation.**command_help**()

Show help for analysis command.

padmet.utils.management.manual_curation.**manual_curation_cli**(*command_args*)

padmet.utils.management.manual_curation.**rxn_creator**(*data_file*, *padmetSpec*, *output*, *padmetRef=None*, *source=None*, *tool=None*, *category='MANUAL'*, *verbose=False*)

Read a data_file (form created with template_new_rxn and filed), for each reaction to create, add the reaction in padmetSpec (only if the id of the reaction is not already in padmetSpec or in padmetRef if given) the source ensure the traceability of the reaction, its a simple tag ex 'pathway_XX_update' if not given the filename of data_file will be used. if a tool was used to infer the reaction, define tool='name_of_the_tool' the Padmet of reference padmetRef can be used to check that the reaction id is not already in the database and copy information from the database for existing compounds strongly recommended to give a padmetRef.

Parameters

- **data_file** (*str*) – path to file based on template_new_rxn()
- **padmetSpec** (*padmet.classes.PadmetSpec*) – padmet to update
- **output** (*str*) – path to the new padmet file
- **source** (*str*) – tag associated to the new reactions to create and add, used for traceability
- **tool** (*str*) – The eventual tool used to infer the reactions to create and add
- **category** (*str*) – The default category of the reaction added manually is 'MANUAL'. Must not be changed.
- **padmetRef** (*padmet.classes.PadmetRef*) – padmet containing the database of reference
- **verbose** (*bool*) – if True print information

padmet.utils.management.manual_curation.**sniff_datafile**(*data_file*)

Read data_file and check which kind of data input it is. A reaction_creator file contains only 2 columns. Add reaction_add_delete more than 2. Basic, need to be improved.

Parameters **data_file** (*str*) – path to file of reaction_creator or reaction_add_delete.

Returns "rxn_creator" or "add_delete_rxn"

Return type str

padmet.utils.management.manual_curation.**template_add_delete**(*output*)
Generate template file used as input of add_delete_rxn function

Parameters **output** (*str*) – path for the template rxn_add_delete to create

padmet.utils.management.manual_curation.**template_new_rxn**(*output*)
Generate template file used as input of rxn_creator function

Parameters **output** (*str*) – path for the template new_rxn to create

padmet_compart

Description: For a given padmet file, check and update compartment.

1./ Get all compartment with 1st usage

2./ Remove a compartment with 2nd usage. Remove all reactions acting in the given compartment

3./ change compartment id with 3rd usage

```
usage:
    padmet padmet_compart --padmet=FILE
    padmet padmet_compart --padmet=FILE --remove=STR [--output=FILE] [-v]
    padmet padmet_compart --padmet=FILE --old=STR --new=STR [--output=FILE] [-v]

options:
    -h --help     Show help.
    --padmet=FILE    pathname of the padmet file
    --remove=STR    compartment id to remove
    --old=STR    compartment id to change to new id
    --new=STR    new compartment id
    --output=FILE    new padmet pathname, if none, overwritting the original padmet
    -v   print info
```

padmet.utils.management.padmet_compart.**command_help**()
Show help for analysis command.

padmet.utils.management.padmet_compart.**padmet_compart_cli**(*command_args*)

padmet.utils.management.padmet_compart.**remove_compart**(*padmet*, *to_remove*, *verbose=False*)
Remove all reaction associated to a compound in the compartment to remove.

Parameters

- **padmet** (*padmet.classes.PadmetSpec*) – padmet to udpate
- **to_remove** (*str*) – compartment id to remove, if many separate compartment id by ','
- **verbose** (*bool*) – if True print information

Returns New padmet after removing compartment(s)

Return type padmet.classes.PadmetSpec

padmet.utils.management.padmet_compart.**replace_compart**(*padmet*, *old_compart*, *new_compart*, *verbose=False*)
Replace compartment 'old_compart' by 'new_compart'.

Parameters

- **padmet** (*padmet.classes.PadmetSpec*) – padmet to udpate
- **old_comaprt** (*str*) – compartment id to remplace
- **new_compart** (*str*) – new compartment id
- **verbose** (*bool*) – if True print information

Returns New padmet after remplacing compartment

Return type padmet.classes.PadmetSpec

padmet_medium

Description: For a given set of compounds representing the growth medium (or seeds). Create 2 reactions for each compounds to maintain consistency of the network for flux analysis. For each compounds create:

An exchange reaction: this reaction consumes the compound in the compartment 'C-BOUNDARY' and produces the compound in the compartment 'e' extracellular

A transport reaction: this reaction consumes the compound in the compartment 'e' extracellular' and produces the compound in the compartment 'c' cytosol ex: for seed 'cpd-a'

1/ check if cpd-a in padmetSpec, if not, copy from padmetRef.

2/ create exchange reaction: ExchangeSeed_cpd-a_b: 1 cpd-a (C-BOUNDARAY) <=> 1 cpd-a (e)

3/ create transport reaction: TransportSeed_cpd-a_e: 1 cpd-a (e) => 1 cpd-a (c)

4/ create a new file if output not None, or overwrite padmetSpec

```
usage:
    padmet padmet_medium --padmetSpec=FILE
    padmet padmet_medium --padmetSpec=FILE -r [--output=FILE] [-v]
    padmet padmet_medium --padmetSpec=FILE --seeds=FILE [--padmetRef=FILE] [--
↪output=FILE] [-v]

options:
    -h --help     Show help.
    --padmetSpec=FILE    path to the padmet file to update
    --padmetRef=FILE    path to the padmet file representing to the database of␣
↪reference (ex: metacyc_18.5.padmet)
    --seeds=FILE    the path to the file containing the compounds ids and the compart,
↪ line = cpd-id        compart.
    --output=FILE    If not None, pathname to the padmet file updated
    -r    Use to remove all medium from padmet
    -v   print info
```

padmet.utils.management.padmet_medium.**command_help**()
Show help for analysis command.

padmet.utils.management.padmet_medium.**manage_medium**(*padmet*, *new_growth_medium=None*, *padmetRef=None*, *verbose=False*)
Manage medium of a padmet. If new_growth_medium give, use this list of compound to define the new medium and create transport and exchange reactions. if padmetRef given, use the information from padmetRef to create the missing compound. If no new_growth_medium given: remove the current medium in the padmet.

Parameters

- **padmet** (*padmet.classes.PadmetSpec*) – padmet to update
- **new_growth_medium** (*list*) – list of compound id representing the medium
- **padmetRef** (*padmet.classes.PadmetRef*) – padmet containing the database of reference
- **verbose** (*bool*) – if True print information

Returns New padmet after updating medium

Return type padmet.classes.PadmetSpec

padmet.utils.management.padmet_medium.**padmet_medium_cli**(*command_args*)

relation_curation

:

usage: padmet relation_curation –padmet=FILE –id_in=STR [–type=STR] [-v] padmet relation_curation –padmet=FILE –id_out=STR [–type=STR] [-v] padmet relation_curation –padmet=FILE –id_in=STR [–type=STR] –to-remove==STR –output=FILE [-v] padmet relation_curation –padmet=FILE –id_in=STR –id_out=STR [–type=STR] –to-remove==STR –output=FILE [-v]

option: -h –help Show help. –model_metabolic=FILE pathname to the metabolic network of the model (sbml). –study_metabolic=FILE **. **–inp=FILE** **. –omcl=FILE **. **–output=FILE** **. -v print info.

padmet.utils.management.relation_curation.**command_help**()
Show help for analysis command.

padmet.utils.management.relation_curation.**get_relations**(*padmet*, *id_in=None*, *id_out=None*, *_type=None*, *to_remove=None*, *output=None*, *verbose=False*)

padmet.utils.management.relation_curation.**relation_curation_cli**(*command_args*)

CHAPTER 6

Getting help

If you've run into issues, found a bug, or can't seem to find an answer to your question regarding the use and functionality of padmet, please use Github Issues page to ask your question.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

D

E

F

G

H

R

S

T

U

V

W

X