semtk-python3

Release 0.1.0

GE Research

Nov 29, 2022

CONTENTS

Welcome to the documentation for semtk-python3!

CHAPTER

ONE

SEMTK3 PACKAGE

semtk3.**build_connection_str**(*name: str*, *triple_store_type: str*, *triple_store_url: str*, *model_graphs: List[str]*, *data_graph: str*, *extra_data_graphs: List[str] = []*)

Throw exception if connection triplestore(s) don't respond OK to http GET

Parameters
conn_str – a SemTK connection json string

semtk3.**build_constraint**(*sparql_id*, *operator*, *operand_list*)

Build a contraint to be used as a query parameter

Parameters

- **sparql_id** – the variable name
- **operator** – operator {MATCHES, REGEX, GREATERTHAN, GREATERTHANOREQUALS, LESSTHAN, LESSTHANOREQUALS, VALUEBETWEEN, VALUEBETWEENUNINCLUSIVE}
- **operand_list** – list of values

Returns
the constraint

Return type
RuntimeConstraint

semtk3.**build_default_connection_str**(*name*, *triple_store_type*, *triple_store_url*)

Build a connection to the default graph only

Parameters

- **name** – name is for display only
- **triple_store_type** – "fuseki" "neptune" "virtuoso", etc.
- **triple_store_url** – the URL e.g. "http://localhost:3030/DATASET"

semtk3.**check_connection_up**(*conn_str*)

Throw exception if connection triplestore(s) don't respond OK to http GET

Parameters
conn_str – a SemTK connection json string

semtk3.**check_services**()

Logs success or failure of each service

Returns
did all pings succeed

Return type
boolean

semtk3.**clear_graph**(*conn_json_str*, *model_or_data*, *index*)

Clear a graph

Parameters

- **conn_json_str** – connection json as a string
- **model_or_data** – string "model" or "data"
- **index** – integer specifying which model or data graph to use

Returns
message

Return type
string

semtk3.**combine_entities**(*target_uri*, *duplicate_uri*, *delete_predicates_from_target=None*, *delete_predicates_from_duplicate=None*, *conn=None*)

Combine two entities. Exception on failure.

Parameters

- **target_uri** – target instance to be combined INTO
- **duplicate_uri** – duplicate instance to be combined then removed
- **delete_predicates_from_target** – list of predicate URIs to be deleted from target
- **delete_predicates_from_duplicate** – list of predicate URIs to be deleted from duplicate
- **conn** – connection (can also be set with set_connection_override())

semtk3.**combine_entities_in_conn**(*same_as_class_uri=None*, *target_prop_uri=None*, *duplicate_prop_uri=None*, *delete_predicates_from_target=[]*, *delete_predicates_from_duplicate=[]*, *conn=None*)

Combine entities described by SameAs instances in the data. See EntityResolution.sadl and Wiki on Entity Resolution

Every param is an unusual override, except perhaps conn. Normal: combine_entities_in_conn() :param _sphinx_paramlinks_semtk3.combine_entities_in_conn.same_as_class_uri: override :param _sphinx_paramlinks_semtk3.combine_entities_in_conn.target_prop_uri: override :param _sphinx_paramlinks_semtk3.combine_entities_in_conn.duplicate_prop_uri: override :param _sphinx_paramlinks_semtk3.combine_entities_in_conn.delete_predicates_from_target: list of propertyURI's to remove from target before combining :param _sphinx_paramlinks_semtk3.combine_entities_in_conn.delete_predicatges_from_duplicate: list of propertyURI's to remove from duplicate before combining :param _sphinx_paramlinks_semtk3.combine_entities_in_conn.conn : connection :return status string :throws exception with table of errors

semtk3.**combine_entities_table**(*csv_str*, *target_col_prop_dict*, *duplicate_col_prop_dict*, *delete_predicates_from_target=[]*, *delete_predicates_from_duplicate=[]*, *conn=None*)

Combine entities described by rows in a table. Each row has col(s) to lookup the target and duplicate Any properties outgoing from duplicate are ignored if they exist in the target All incoming properties are combined delete_predicates_from_* parameters occur before combining using the above rules

"#type" may be used as a property shorthand

Parameters

- **csv_str** – csv table of entities to combine
- **target_col_prop_dict** – dictionary describing how to look up target dict[col_name]=prop_uri
- **duplicate_col_prop_dict** – dictionary describing how to look up duplicate dict[col_name]=prop_uri
- **delete_predicates_from_target** – list of propertyURI's to remove from target before combining
- **delete_predicatges_from_duplicate** – list of propertyURI's to remove from duplicate before combining
- **conn** – connection

Returns
status string

Throws
exception with table of errors

semtk3.**copy_graph**(*from_graph: str*, *to_graph: str*, *from_server: Optional[str] = None*, *from_server_type: Optional[str] = None*, *to_server: Optional[str] = None*, *to_server_type: Optional[str] = None*, *user_name='noone'*, *password='nopass'*)

Copy one graph to another (merging into the destination) So clear the "to" graph as a separate step if desired.

Parameters

- **from_graph** – merge from this graph
- **to_graph** – merge to this graph
- **from_server** – merge from this server. if None: get from SEMTK_CONN_OVERRIDE.data[0]
- **from_server_type** – type of 'from' server. if None: get from SEMTK_CONN_OVERRIDE.data[0]
- **to_server** – merge to this server. if None: get from SEMTK_CONN_OVERRIDE.data[0]
- **to_server_type** – type of 'to' server. if None: get from SEMTK_CONN_OVERRIDE.data[0]
- **user_name** – if security needed on 'to' server
- **password** – if security needed on 'to' server

Returns
status message string like "successfully copied uri://from into uri://to

Throws
exception on any error

semtk3.**count_by_id**(*nodegroup_id*, *limit_override=0*, *offset_override=0*, *runtime_constraints=None*, *edc_constraints=None*, *flags=None*)

Execute a count query for a given nodegroup id

Parameters

- **nodegroup_id** – id of nodegroup in the store
- **limit_override** – optional override of LIMIT clause

- **offset_override** – optional override of OFFSET clause
- **runtime_constraints** – optional runtime constraints built by build_constraint()
- **edc_constraints** – optional edc constraints
- **flags** – optional query flags

Returns
results

Return type
semtktable

semtk3.**create_nodegroup**(*conn_json_str*, *class_uri*, *sparql_id=None*)

Create a nodegroup containing a single uri

Parameters

- **conn_json_str** – connection json string
- **class_uri** – class to add
- **sparql_id** – optional sparqlID if different from ?ClassName

Returns
nodegroup

Return type
nodegroup json string

semtk3.**delete_item_from_store**(*item_id*, *item_type*)

Delete item from the store, error if it doesn't exist.

Parameters

- **item_id** – the id
- **item_type** – one of STORE_ITEM_TYPE

semtk3.**delete_items_from_store**(*regex_str*, *item_type='StoredItem'*)

Delete matching nodegroups from store

Parameters

- **regex_str** – pattern to search() on nodegroup id's (any match in id)
- **item_type** – only delete items of this type

semtk3.**delete_nodegroup_from_store**(*nodegroup_id*)

Delete nodegroup_id from the store error if it doesn't exist.

Parameters
nodegroup_id – the id

semtk3.**delete_nodegroups_from_store**(*regex_str*)

semtk3.**download_owl**(*owl_file_path*, *conn_json_str*, *user_name='noone'*, *password='nopass'*, *model_or_data='model'*, *conn_index=0*)

Download a graph as an OWL file

Parameters

- **owl_file_path** – path to the file

- **conn_json_str** – connection json string (defaults to the first MODEL graph in the connection)
- **user_name** – optional user name
- **password** – optional password
- **model_or_data** – optional "model" or "data" specifying which endpoint in the sparql connection, defaults to "data"
- **conn_index** – index specifying which of the model or data endpoints in the sparql connection, defaults to 0

Returns
None - raises exception on error

semtk3.**fdc_cache_bootstrap_table**(*conn_json_str*, *spec_id*, *bootstrap_table*, *recache_after_sec*)

Run an fdc cache spec

Parameters

- **conn_json_str** – connection containing model and data graphs
- **spec_id** – the fdc cache spec identifier
- **bootstrap_table** – semtktable to kick off the cache
- **recache_after_sec** – maximum age of cache

semtk3.**get_class_names**(*conn_json_str=None*)

Get a list of class names in the ontology

Parameters
conn_json_str – optional conenction json string defaults to override

:returns list of full class URI's

semtk3.**get_class_template**(*class_uri*, *conn_json_str=None*, *id_regex='identifier'*)

Get class template nodegroup

Parameters

- **class_uri** – the class whose template should be used for ingestion
- **conn_json_str** – optional conenction json string defaults to override
- **id_regex** – optional regex to identify the key data properties of classes which are the object of object properties

:returns nodegroup json string

semtk3.**get_class_template_and_csv**(*class_uri*, *conn_json_str=None*, *id_regex='identifier'*)

Get class template nodegroup

param class_uri
the class whose template should be used for ingestion

param conn_json_str
optional conenction json string defaults to override

param id_regex
optional regex to identify the key data properties of classes which are the object of object properties

:returns (ng_json_str, "col1, col2, col3

", "string, int, dateTime ") note that types can be space-separated complex property types

semtk3.**get_class_template_csv**(*class_uri*, *conn_json_str=None*, *id_regex='identifier'*)

Get sample CSV that will work with class template

Parameters

- **class_uri** – the class whose template should be used for ingestion
- **conn_json_str** – optional conenction json string defaults to override
- **id_regex** – optional regex to identify the key data properties of classes which are the object of object properties

Returns

"colname1, colname2, colname3"

semtk3.**get_constraints_by_id**(*nodegroup_id*)

Get runtime constraints for a stored nodegroup

Parameters

nodegroup_id – the id

Returns

columns valueId, itemType and valueType

Return type

semtktable

semtk3.**get_filter_values_by_id**(*nodegroup_id*, *target_obj_sparql_id*, *override_conn_json_str=None*, *limit_override=None*, *offset_override=None*, *runtime_constraints=None*, *edc_constraints=None*, *flags=None*)

Run a filter values query, which returns all the existing values for a given variable in the nodegroup

Parameters

- **nodegroup_id** – the id
- **target_obj_sparql_id** – the variable to be interrogated
- **override_conn_json_str** – optional override connection json string
- **limit_override** – optional override of LIMIT clause
- **offset_override** – optional override of OFFSET clause
- **runtime_constraints** – optional runtime constraints built by build_constraint()
- **edc_constraints** – optional edc constraints
- **flags** – optional query flags

Returns

results

Return type

semtktable

semtk3.**get_graph_info**(*conn_json_str*, *skip_semtk_graphs=False*, *graph_names_only=True*)

Get names and triple counts of graphs present in the triple store

Parameters

- **conn_json_str** – connection json string
- **skip_semtk_graphs** – true to exclude SemTK utility graphs

- **graph_names_only** – true to only return graph names. False to return other info like triple counts.

Returns
a table with graph names and (optionally) triple counts

Return type
semtktable

semtk3.**get_instance_dictionary**(*max_words: int = 2*, *specificity_limit: int = 1*, *conn_json_str: Optional[str] = None*) → *SemtkTable*

Get a table describing the uris and their labels. Columns:

- instance_uri - the URI
- class_uris - instance belongs to one or more classes
- label - label (or name) associated with the instance. NOT UNIQUE: see label_specificity
- label_specificity - how many uris have this label
- property - what prop was used to associate label with instance_uri

Parameters

- **max_words** – the maximum number of words a string may have and be considered a label
- **specificity_limit** – the maximum number of URI's one-hop from the label for it to be returned
- **conn_json_str** – connection string of graph(s) holding the model

Return type
semtktable

semtk3.**get_logger**()

semtk3.**get_nodegroup_by_id**(*nodegroup_id*)

Retrieve a nodegroup from the store

Parameters
nodegroup_id – the id

Returns
a nodegroup

Return type
json string

semtk3.**get_nodegroup_store_data**()

Get list of nodegroups in the nodegroup store

Returns
SemtkTable with columns 'ID', 'comments', 'creationDate', 'creator', 'itemType'

Return type
semtktable

semtk3.**get_oinfo**(*conn_json_str=None*)

Get a table describing the ontology model

Parameters
conn_json_str – connection string of graph(s) holding the model

Return type
semtktable

semtk3.**get_oinfo_predicate_stats**(*conn_json_str=None*)

Get a table describing the ontology model

Parameters
conn_json_str – connection string of graph(s) holding the model

Return type
semtktable

semtk3.**get_oinfo_uri_label_table**(*conn_json_str=None*)

Get a table describing the ontology model

Parameters
conn_json_str – connection string of graph(s) holding the model

Return type
semtktable

semtk3.**get_plot_spec_names_by_id**(*nodegroup_id*)

Get available plot names for a given nodegroup id

semtk3.**get_sparqlgraph_url**(*host_url: str*, *nodegroup_id: Optional[str] = None*, *report_id: Optional[str] = None*, *runtime_constraints: Optional[List[RuntimeConstraint]] = None*, *run_flag: Optional[str] = None*, *conn_json_str: Optional[str] = None*)

Get a URL for sparqlgraph with params to launch a connection, nodegroup, query

Parameters

- **host_url** (*str*) – base url e.g. http://localhost:8080
- **nodegroup_id** (*str*) – id of nodegroup in the store to launch. By default, run the query.
- **report_id** (*str*) – id of report in the store to launch. By default, run the report.
- **runtime_constraints** (RuntimeConstraint) – constraints to apply to query if nodegroup_id is specified
- **run_flag** (*str*) – "True" or "False", default "True"
- **conn_json_str** (*str*) – connection to load. Will override nodegroup_id's.

Returns
url

Return type
string

semtk3.**get_store_item**(*item_id*, *item_type*)

semtk3.**get_store_table**(*item_type='StoredItem'*)

Get list of everything in the store

Parameters
item_type – one of the STORE_ITEM_TYPE constants

Returns
SemtkTable with columns 'ID', 'comments', 'creationDate', 'creator', 'itemType'

Return type
semtktable

semtk3.**get_table**(*jobid*)

Get a table from an async job

Parameters
jobid – the job id

Return type
semtktable

semtk3.**ingest_by_id**(*nodegroup_id*, *csv_str*, *override_conn_json_str=None*)

Perform data ingestion, throwing exception on failure

Parameters

- **nodegroup_id** – nodegroup with ingestion template
- **csv_str** – string csv data
- **override_conn_json_str** – optional override connection

Returns
(statusMsg, warnMsg) where warnMsg is often ‘’

Return type
string tuple

semtk3.**ingest_using_class_template**(*class_uri*, *csv_str*, *conn_json_str=None*, *id_regex='identifier'*)

Ingest using class template, throwing exception on failure

Parameters

- **class_uri** – the class whose template should be used for ingestion
- **csv_str** – string csv data
- **id_regex** – regex matching properties that should be used for lookups

Conn_json_str
connection

Returns
(statusMsg, warnMsg) where warnMsg is often ‘’

Return type
string tuple

semtk3.**main**()

semtk3.**override_hosts**(*query_host=None*, *status_host=None*, *results_host=None*, *hive_host=None*, *oinfo_host=None*, *nodegroup_exec_host=None*, *nodegroup_host=None*, *utility_host=None*, *fdcache_host=None*, *ingestion_host=None*)

Override the default host(s) for Semtk service(s).

Parameters

- **query_host** – optional
- **status_host** – optional
- **results_host** – optional
- **hive_host** – deprecated
- **oinfo_host** – optional
- **nodegroup_exec_host** – optional

- **nodegroup_host** – optional
- **fdcache_host** – optional
- **ingestion_host** – optional

semtk3.**override_ports**(*query_port=None*, *status_port=None*, *results_port=None*, *hive_port=None*, *oinfo_port=None*, *nodegroup_exec_port=None*, *nodegroup_port=None*, *utility_port=None*, *fdcache_port=None*, *ingestion_port=None*)

Override the default port(s) for Semtk service(s). Ports may be numbers (port will be appended with colon), e.g. 80 or "80" or context string (port will simply be appended) e.g. "/query"

Parameters

- **query_port** – optional
- **status_port** – optional
- **results_port** – optional
- **hive_port** – deprecated
- **oinfo_port** – optional
- **nodegroup_exec_port** – optional
- **nodegroup_port** – optional
- **fdcache_port** – optional
- **ingestion_port** – optional

semtk3.**print_wait_dots**(*seconds*)

semtk3.**query**(*query*, *conn_json_str*, *model_or_data='data'*, *conn_index=0*)

Run a raw SPARQL query

Parameters

- **query** – SPARQL
- **conn_json_str** – connection json string
- **model_or_data** – optional "model" or "data" specifying which endpoint in the sparql connection, defaults to "data"
- **conn_index** – index specifying which of the model or data endpoints in the sparql connection, defaults to 0

Returns
results

Return type
semtktable

semtk3.**query_by_id**(*nodegroup_id*, *limit_override=0*, *offset_override=0*, *runtime_constraints=None*, *edc_constraints=None*, *flags=None*, *query_type=None*, *result_type=None*)

Execute the default query type for a given nodegroup id

Check results for type(result) is

dict - json ld results

semtk3.semtktable.SemtkTable

A count query will be a SemtkTable with colum nname "count"

A confirm query will be a SemtkTable with column name "@message"

Parameters

- **nodegroup_id** – id of nodegroup in the store
- **limit_override** – optional override of LIMIT clause
- **offset_override** – optional override of OFFSET clause
- **runtime_constraints** – optional runtime constraints built by build_constraint()
- **edc_constraints** – optional edc constraints
- **flags** – optional query flags

Returns
results: dict or semtk3.semtktable.SemtkTable

Return type
semtktable or JSON

semtk3.**query_by_nodegroup**(*nodegroup_str*, *runtime_constraints=None*, *edc_constraints=None*, *flags=None*, *query_type=None*, *result_type=None*)

Execute the default query type for a given nodegroup id

Check results for type(result) is

dict - json ld results

semtk3.semtktable.SemtkTable

A count query will be a SemtkTable with colum nname "count"

A confirm query will be a SemtkTable with column name "@message"

Parameters

- **nodegroup_str** – nodegroup
- **runtime_constraints** – optional runtime constraints built by build_constraint()
- **edc_constraints** – optional edc constraints
- **flags** – optional query flags

Returns
results: dict or semtk3.semtktable.SemtkTable

Return type
semtktable or JSON

semtk3.**query_hive**(*hiveserver_host*, *hiveserver_port*, *hiveserver_database*, *query*)

semtk3.**retrieve_from_store**(*regex_str*, *folder_path*)

semtk3.**retrieve_items_from_store**(*regex_str*, *folder_path*, *item_type='StoredItem'*)

Retrieve all items matching a pattern, create store_data.csv

Parameters

- **regex_str** – pattern to match on nodegroup id's

- **folder_path** – target folder

semtk3.**retrieve_nodegroups_from_store**(*regex_str*, *folder_path*)

semtk3.**retrieve_reports_from_store**(*regex_str*, *folder_path*)

Retrieve all items matching a pattern, create store_data.csv Retrieves reports and any nodegroups they use

Parameters

- **regex_str** – pattern to match on nodegroup id's
- **folder_path** – target folder

semtk3.**select_by_id**(*nodegroup_id*, *limit_override=0*, *offset_override=0*, *runtime_constraints=None*, *edc_constraints=None*, *flags=None*)

Execute a select query for a given nodegroup id

Parameters

- **nodegroup_id** – id of nodegroup in the store
- **limit_override** – optional override of LIMIT clause
- **offset_override** – optional override of OFFSET clause
- **runtime_constraints** – optional runtime constraints built by build_constraint()
- **edc_constraints** – optional edc constraints
- **flags** – optional query flags

Returns
results

Return type
semtktable

semtk3.**select_plot_by_id**(*nodegroup_id*, *plot_name*)

Create a plot for a given nodegroup id

semtk3.**set_connection_override**(*conn_str*)

Set a connection string to be used in all nodegroups

Parameters
conn_str – a SemTK connection json string

semtk3.**set_headers**(*headers*)

semtk3.**set_host**(*hostUrl*)

semtk3.**store_folder**(*folder_path*)

Reads a file of the standard "store_data.csv" format
ID,comments,creator,jsonfile, optional: type id27,Test comments,200001111,file.json

…and saves the specified nodegroups to the store, overwriting existing if needed

Parameters
folder_path – target folder

semtk3.**store_item**(*item_id*, *comments*, *creator*, *item_json_str*, *item_type*, *overwrite_flag=False*)

Saves a single item to the store, fails if nodegroup_id already exists unless overwrite_flag

Parameters

- **item_id** – the id
- **comments** – comment string
- **creator** – creator string
- **item_json_str** – json string of NODEGROUP or REPORT, etc.
- **item_type** – one of the STORE_ITEM_TYPE constants
- **overwrite_flag** – if true then silently overwrite existing item with the same name

Returns
status

Return type
string

semtk3.**store_nodegroup**(*nodegroup_id*, *comments*, *creator*, *nodegroup_json_str*, *overwrite_flag=False*)

Saves a single nodegroup to the store, fails if nodegroup_id already exists unless overwrite_flag

Parameters

- **nodegroup_id** – the id
- **comments** – comment string
- **creator** – creator string
- **nodegroup_json_str** – nodegroup in json string form

Returns
status

Return type
string

semtk3.**store_nodegroups**(*folder_path*)

semtk3.**upload_owl**(*owl_file_path*, *conn_json_str*, *user_name='noone'*, *password='nopass'*, *model_or_data='model'*, *conn_index=0*)

Upload an owl file to a given graph

Parameters

- **owl_file_path** – path to the file
- **conn_json_str** – connection json string
- **user_name** – optional user name
- **password** – optional password
- **model_or_data** – optional “model” or “data” specifying which graph in the sparql connection, defaults to “model”
- **conn_index** – index specifying which of the model or data graphs in the sparql connection, defaults to 0

Returns
message

Return type
string

semtk3.**upload_turtle**(*ttl_file_path*, *conn_json_str*, *user_name*, *password*, *model_or_data='model'*, *conn_index=0*)

Upload an turtle file

Parameters

- **ttl_file_path** – path to the file
- **conn_json_str** – connection json string
- **user_name** – optional user name
- **password** – optional password
- **model_or_data** – optional "model" or "data" specifying which endpoint in the sparql connection, defaults to "model"
- **conn_index** – index specifying which of the model or data endpoints in the sparql connection, defaults to 0

Returns
message

Return type
string

1.1 Submodules

1.1.1 semtk3.clients module

Created on May 6, 2019

@author: 200001934

semtk3.clients.**foo**()

1.1.2 semtk3.demo module

1.1.3 semtk3.edcclient module

class semtk3.edcclient.**EdcClient**(*baseURL*, *service=None*, *status_client=None*, *results_client=None*)

Bases: *SemTkAsyncClient*

post_async_to_table(*endpoint*, *dataObj={}*)

returns SemTkTable raises errors otherwise

post_edc_to_table(*endpoint*, *dataObj={}*)

post_to_table(*endpoint*, *dataObj={}*)

returns dict - the table raises RestException

1.1.4 semtk3.fdccacheclient module

class semtk3.fdccacheclient.**FdcCacheClient**(*serverURL*, *status_client=None*, *results_client=None*)

Bases: *SemTkAsyncClient*

exec_cache_using_table_bootstrap(*conn_json_str*, *spec_id*, *bootstrap_table*, *recache_after_sec*)

1.1.5 semtk3.nodegroupclient module

class semtk3.nodegroupclient.**NodegroupClient**(*serverURL*, *status_client*, *results_client*)

Bases: *SemTkAsyncClient*

USE_NODEGROUP_CONN = '{"name": "%NODEGROUP%","domain": "%NODEGROUP%","model": [],"data": []}'

exec_create_nodegroup(*conn_json_str*, *class_uri*, *sparql_id=None*)

execute a create_nodegroup throws: exception otherwise

1.1.6 semtk3.nodegroupexecclient module

class semtk3.nodegroupexecclient.**NodegroupExecClient**(*serverURL*, *status_client=None*, *results_client=None*)

Bases: *SemTkAsyncClient*

USE_NODEGROUP_CONN = '{"name": "%NODEGROUP%","domain": "%NODEGROUP%","model": [],"data": []}'

exec_async_dispatch_count_by_id(*nodegroup_id*, *override_conn_json_str=None*, *limit_override=None*, *offset_override=None*, *runtime_constraints=None*, *edc_constraints=None*, *flags=None*)

execute a count by nodegroup id returns: the table thorws: exception otherwise

exec_async_dispatch_filter_by_id(*nodegroup_id*, *target_obj_sparql_id*, *override_conn_json_str=None*, *limit_override=None*, *offset_override=None*, *runtime_constraints=None*, *edc_constraints=None*, *flags=None*)

execute a select by nodegroup id returns: the table thorws: exception otherwise

exec_async_dispatch_query_by_id(*nodegroup_id*, *override_conn_json_str=None*, *limit_override=None*, *offset_override=None*, *runtime_constraints=None*, *edc_constraints=None*, *flags=None*, *query_type=None*, *result_type=None*)

execute default query type nodegroup id returns: One of: table, json, integer thorws: exception otherwise

exec_async_dispatch_query_from_nodegroup(*nodegroup_str*, *override_conn_json_str=None*, *runtime_constraints=None*, *edc_constraints=None*, *flags=None*, *query_type=None*, *result_type=None*)

execute default query type nodegroup id returns: One of: table, json, integer thorws: exception otherwise

exec_async_dispatch_raw_sparql(*sparql*, *override_conn_json_str=None*)

execute a select by nodegroup id returns: the table thorws: exception otherwise

exec_async_dispatch_select_by_id(*nodegroup_id*, *override_conn_json_str=None*, *limit_override=None*, *offset_override=None*, *runtime_constraints=None*, *edc_constraints=None*, *flags=None*)

execute a select by nodegroup id returns: the table thorws: exception otherwise

exec_async_ingest_from_csv(*nodegroup_id*, *csv_str*, *override_conn_json_str=None*)

nodegroup_id - from nodegroup store csv_str - data, e.g. from: open('data.csv', 'r').read() override_conn_json_str - string with json of a different connection

returns status,warnings where status is always a string and warnings might be ""

exec_copy_graph(*from_server*, *from_server_type*, *from_graph*, *to_server*, *to_server_type*, *to_graph*, *user_name='no_user'*, *password='no_password'*)

exec_dispatch_clear_graph(*conn*, *model_or_data*, *index*)

execute clear graph returns: message,

some text

throws: exception otherwise

exec_dispatch_combine_entities(*target_uri*, *duplicate_uri*, *delete_predicates_from_target*, *delete_predicates_from_duplicate*, *conn_json_str*)

exec_dispatch_combine_entities_in_conn(*same_as_class_uri*, *target_prop_uri*, *duplicate_prop_uri*, *delete_predicates_from_target*, *delete_predicates_from_duplicate*, *conn_json_str*)

exec_dispatch_combine_entities_table(*csv_str*, *target_col_prop_dict*, *duplicate_col_prop_dict*, *delete_predicates_from_target*, *delete_predicates_from_duplicate*, *conn_json_str*)

exec_get_runtime_constraints_by_id(*nodegroup_id*)

execute a select by nodegroup id returns: valueId, itemType, valueType

?sparqlId PROPERTYITEM INT|STRING|FLOAT etc. ?sparqlId2 NODE NODE_URI

throws: exception otherwise

1.1.7 semtk3.nodegroupstoreclient module

class semtk3.nodegroupstoreclient.**NodegroupStoreClient**(*serverURL*)

Bases: *SemTkClient*

exec_delete_stored_item(*item_id*, *item_type*)

exec_get_stored_item_by_id(*item_id*, *item_type*)

exec_get_stored_items_metadata(*item_type*)

exec_store_item(*item_id*, *comments*, *creator*, *item_json_str*, *item_type*)

1.1.8 semtk3.oinfoclient module

class semtk3.oinfoclient.**OInfoClient**(*serverURL*, *conn_json_str*, *status_client=None*, *results_client=None*)

Bases: *SemTkAsyncClient*

exec_get_instance_dictionary(*max_words: int = 2*, *specificity_limit: int = 1*)

exec_get_ontology_info()

exec_get_predicate_stats()

exec_get_uri_label_table()

1.1.9 semtk3.queryclient module

class semtk3.queryclient.**QueryClient**(*serverURL*, *conn_obj*)

Bases: *SemTkClient*

exec_download_owl(*owl_file_path*, *model_or_data='model'*, *index=0*)

exec_query(*query*, *model_or_data='data'*, *index=0*)

exec_select_graph_names(*skip_semtk_graphs*, *graph_names_only*)

exec_upload_owl(*owl_file_path*, *model_or_data='model'*, *index=0*)

exec_upload_turtle(*turtle_file_path*, *model_or_data='model'*, *index=0*)

1.1.10 semtk3.restclient module

class semtk3.restclient.**RestClient**(*baseURL*, *service=None*)

Bases: `object`

HEADERS = {'cache-control': 'no-cache'}

post(*endpoint*, *dataObj={}*, *files=None*)

basic POST endpoint - string dataObj - dict will be converted to json for the post (default {})

returns string - response content raises RestException if response is not OK

post_to_file(*endpoint*, *dataObj*, *filename*)

raise_exception(*msg*)

static set_headers(*headers*)

to_json_array(*to_jsonable_list*)

exception semtk3.restclient.**RestException**

Bases: `Exception`

Exception for errors from rest endpoints

1.1.11 semtk3.resultsclient module

class semtk3.resultsclient.**ResultsClient**(*serverURL*)

Bases: *SemTkClient*

exec_get_binary_file(*fileId*, *baseDir*)

Download file into baseDir return the path

exec_get_json_blob_results(*jobId*)

returns SemtkTable

exec_get_table_results(*jobId*)

returns SemtkTable

1.1.12 semtk3.runtimeconstraint module

class semtk3.runtimeconstraint.**RuntimeConstraint**(*sparqlId*, *operator*, *operand_list*)

Bases: object

OP_GREATERTHAN = 'GREATERTHAN'

OP_GREATERTHANOREQUALS = 'GREATERTHANOREQUALS'

OP_LESSTHAN = 'LESSTHAN'

OP_LESSTHANOREQUALS = 'LESSTHANOREQUALS'

OP_MATCHES = 'MATCHES'

OP_NOTMATCHES = 'NOTMATCHES'

OP_REGEX = 'REGEX'

OP_VALUEBETWEEN = 'VALUEBETWEEN'

OP_VALUEBETWEENUNINCLUSIVE = 'VALUEBETWEENUNINCLUSIVE'

to_json()

1.1.13 semtk3.semtkasyncclient module

class semtk3.semtkasyncclient.**SemTkAsyncClient**(*baseURL*, *service=None*, *status_client=None*, *results_client=None*)

Bases: *SemTkClient*

PERCENT_INCREMENT = 20

PRINT_DOTS = False

WAIT_MSEC = 5000

exec_get_job_completion_percentage(*jobid*)

returns int

exec_get_results_table(*jobid*)
returns SemtkTable

exec_job_status_boolean(*jobid*)
returns boolean

exec_job_status_message(*jobid*)
returns string

exec_wait_for_percent_or_msec(*jobid*, *percent_complete*, *max_wait_msec*)
returns integer percent complete

poll_until_success(*jobid*)
poll for percent complete and return if SUCCESS raises RestException including if status="failure"

returns void

post_async_to_json_blob(*endpoint*, *dataObj={}*)
returns json raises errors otherwise

post_async_to_record_process(*endpoint*, *dataObj={}*)
returns success message, which may include warnings raises errors including error table

post_async_to_status(*endpoint*, *dataObj={}*)
returns success message raises errors including status != success

post_async_to_table(*endpoint*, *dataObj={}*)
returns SemTkTable raises errors otherwise

post_get_json_blob_results(*jobid*)
get table results using results, otherwise using self

post_get_percent_complete(*jobid*)
get percent complete using status, otherwise using self

post_get_status_boolean(*jobid*)
get status using status client, otherwise using self

post_get_status_message(*jobid*)
get status message using status client, otherwise using self

post_get_table_results(*jobid*)
get table results using results, otherwise using self

post_wait_for_percent_or_msec(*jobid*, *percent_complete*, *max_wait_msec*)

1.1.14 semtk3.semtkclient module

class semtk3.semtkclient.**SemTkClient**(*baseURL*, *service=None*)
Bases: *RestClient*

JOB_ID_KEY = 'JobId'

RESULT_TYPE_KEY = 'resultType'

WARNINGS_KEY = 'warnings'

get_simple_field(*simple_res*, *field*)

get a simple field with REST error handling

get_simple_field_int(*simple_res*, *field*)

get integer simple results field returns int raises RestException on type or missing field

get_simple_field_str(*simple_res*, *field*)

get string from simple result returns string raises RestException on type or missing field

ping()

logger.INFO(success) or logger.ERROR(error) returns True (success) or False (failure)

post_to_jobid(*endpoint*, *dataObj={}*)

returns string jobid raises errors otherwise

post_to_jobid_warnings(*endpoint*, *dataObj={}*)

for ingestion jobs which return warnings at the initial call returns string jobid, ["warning1", "warning2"] (where warnings can be None) raises errors otherwise

post_to_record_process(*endpoint*, *dataObj={}*, *files=None*)

returns records processed successfully raises RestException unless failuresEncountered = 0

post_to_simple(*endpoint*, *dataObj={}*, *files=None*)

returns dict - the simple results

which can be used as a regular dict, or with error-handling get_simple_field*() methods

post_to_status(*endpoint*, *dataObj={}*, *files=None*)

throws error if status is not success returns dict - the simple results,

which can be used as a regular dict, or with error-handling get_simple_field*() methods

post_to_table(*endpoint*, *dataObj={}*)

returns dict - the table raises RestException

1.1.15 semtk3.semtktable module

class semtk3.semtktable.**SemtkTable**(*table_dict*)

Bases: object

static create_table_dict(*col_names*, *col_types*, *rows*)

delete_column(*col_name*)

get_cell(*row*, *col*)

get_cell_as_date(*row*, *col*)

get_cell_as_float(*row*, *col*)

get_cell_as_int(*row*, *col*)

get_cell_as_string(*row*, *col*)

get_cell_typed(*row*, *col*)

PEC TODO Full types list : especially Time (see list in ImportSpecHandler.java)

get_column(*col*)

get_column_index(*col_name*)

get column index raises ValueError

get_column_names()

get_column_type(*col_name*)

raises ValueError on bad col_name

get_column_types()

get_csv_string()

get_matching_row_nums(*col_name*, *regex_str*)

get_matching_rows(*col_name*, *regex_str*)

get_num_columns()

get_num_rows()

get_pandas_data()

get_rows()

returns array of arrays

has_column(*col_name*)

set_cell(*row*, *col*, *val*)

to_dict()

to_json_str()

1.1.16 semtk3.sparqlconnection module

class semtk3.sparqlconnection.**SparqlConnection**(*conn_json_str='{}'*, *user_name=None*, *password=None*)

Bases: object

DATA = 'data'

MODEL = 'model'

build(*name*, *triple_store_type*, *triple_store*, *model_graphs*, *data_graph*, *extra_data_graphs=[]*)

build a connection @param name : name @param triple_store_type : "fuseki" "neptune" "virtuoso", etc. @param triple_store : the URL e.g. "http://localhost:3030/DATASET" @model_graphs : list of model graphs e.g. ["uri://my_graph", "http://my/other#graph"] @data_graph : default ingestion data graph e.g. "uri://my_graph" @extra_data_graphs : list of data graphs with ["uri://my_graph", "http://my/other#graph"]

get_all_triplestore_urls()

get_graph(*model_or_data*, *index*)

get_password()

get_server_and_port(*model_or_data*, *index*)

get_server_type(*model_or_data*, *index*)

get_user_name()

to_conn_str()

1.1.17 semtk3.statusclient module

class semtk3.statusclient.**StatusClient**(*serverURL*)

Bases: *SemTkClient*

exec_get_percent_complete(*jobId*)

returns int

exec_get_status_boolean(*jobId*)

returns boolean

exec_get_status_message(*jobid*)

returns string

exec_wait_for_percent_or_msec(*jobid*, *percent_complete*, *max_wait_msec*)

returns integer percent complete

1.1.18 semtk3.util module

@author: 200001934

semtk3.util.**download_url**(*url*, *baseDir*)

download a URL contents to baseDir attempt to use filename from headers, else generate a guid append _0 _1 etc to base filename to avoid duplicates

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

F

G

H

I

J

M

N

O

P

Q

R

S

T

U

W