pystardog Documentation

Pedro Oliveira, John Bresnahan, Stephen Nowell

Apr 28, 2021

CONTENTS:

CHAPTER

ONE

PYSTARDOG

Python wrapper for communicating with the Stardog HTTP server.

1.1 What is it?

This framework wraps all the functionality of a client for the Stardog Knowledge Graph, and provides access to a full set of functions such as executing SPARQL queries, administrative tasks on Stardog, and the use of the Reasoning API.

The implementation uses the HTTP protocol, since most of Stardog functionality is available using this protocol. For more information, go to the Stardog's HTTP Programming documentation.

1.2 Installation

pystardog is on PyPI so all you need is: `pip install pystardog`

1.3 Documentation

Documentation is readable at Read the Docs or can be built using Sphinx:

```
pip install -r requirements.txt
cd docs
make html
```

1.4 Tests

Create a virtual environment:

```
virtualenv -p $(which python3) venv
```

Activate the virtualenv:

```
$ . venv/bin/activate
(venv) $
```

Install the dependencies

```
pip install -r requirements.txt
```

Run the tests

```
python setup.py test
```

1.5 Quick Example

```
import stardog

conn_details = {
  'endpoint': 'http://localhost:5820',
  'username': 'admin',
  'password': 'admin'
}

with stardog.Admin(**conn_details) as admin:
  db = admin.new_database('db')

  with stardog.Connection('db', **conn_details) as conn:
    conn.begin()
    conn.add(stardog.content.File('./test/data/example.ttl'))
    conn.commit()
    results = conn.select('select * { ?a ?p ?o }')

  db.drop()
```

1.6 Interactive Tutorial

There is a Jupyter notebook and instructions in the `notebooks` directory of this repository.

CHAPTER

TWO

MODULES

2.1 stardog.connection

Connect to Stardog databases.

class stardog.connection.**Connection**(*database*, *endpoint=None*, *username=None*, *password=None*, *auth=None*)

Bases: object

Database Connection.

This is the entry point for all user-related operations on a Stardog database

__init__(*database*, *endpoint=None*, *username=None*, *password=None*, *auth=None*)

Initializes a connection to a Stardog database.

Parameters

- **database** (*str*) – Name of the database
- **endpoint** (*str*) – Url of the server endpoint. Defaults to *http://localhost:5820*
- **username** (*str, optional*) – Username to use in the connection
- **password** (*str, optional*) – Password to use in the connection
- **auth** (*requests.auth.AuthBase, optional*) – requests Authentication object. Defaults to *None*

Examples

```
>>> conn = Connection('db', endpoint='http://localhost:9999',
                      username='admin', password='admin')
```

add(*content*, *graph_uri=None*)

Adds data to the database.

Parameters

- **content** (Content) – Data to add
- **graph_uri** (*str, optional*) – Named graph into which to add the data

Raises ***stardog.exceptions.TransactionException*** – If not currently in a transaction

Examples

```
>>> conn.add(File('example.ttl'), graph_uri='urn:graph')
```

ask(*query*, ***kwargs*)
Executes a SPARQL ask query.

Parameters

- **query** (*str*) – SPARQL query
- **base_uri** (*str, optional*) – Base URI for the parsing of the query
- **limit** (*int, optional*) – Maximum number of results to return
- **offset** (*int, optional*) – Offset into the result set
- **timeout** (*int, optional*) – Number of ms after which the query should timeout. 0 or less implies no timeout
- **reasoning** (*bool, optional*) – Enable reasoning for the query
- **bindings** (*dict, optional*) – Map between query variables and their values

Returns Result of ask query

Return type bool

Examples

```
>>> conn.ask('ask {:subj :pred :obj}', reasoning=True)
```

begin(***kwargs*)
Begins a transaction.

Parameters **reasoning** (*bool, optional*) – Enable reasoning for all queries inside the transaction. If the transaction does not have reasoning enabled, queries within will not be able to use reasoning.

Returns Transaction ID

Return type str

Raises ***stardog.exceptions.TransactionException*** – If already in a transaction

clear(*graph_uri=None*)
Removes all data from the database or specific named graph.

Parameters **graph_uri** (*str, optional*) – Named graph from which to remove data

Raises ***stardog.exceptions.TransactionException*** – If currently not in a transaction

Examples

clear a specific named graph

```
>>> conn.clear('urn:graph')
```

clear the whole database

```
>>> conn.clear()
```

close()
Close the underlying HTTP connection.

commit()
Commits the current transaction.

Raises ***stardog.exceptions.TransactionException*** – If currently not in a transaction

docs()
Makes a document storage object.

Returns A Docs object

Return type *Docs*

explain(*query*, *base_uri=None*)
Explains the evaluation of a SPARQL query.

Parameters

- **query** (*str*) – SPARQL query
- **base_uri** (*str, optional*) – Base URI for the parsing of the query

Returns Query explanation

Return type str

explain_inconsistency(*graph_uri=None*)
Explains why the database or a named graph is inconsistent.

Parameters **graph_uri** (*str, optional*) – Named graph for which to explain inconsistency

Returns Explanation results

Return type dict

explain_inference(*content*)
Explains the given inference results.

Parameters **content** (Content) – Data from which to provide explanations

Returns Explanation results

Return type dict

Examples

```
>>> conn.explain_inference(File('inferences.ttl'))
```

export(*content_type='text/turtle'*, *stream=False*, *chunk_size=10240*, *graph_uri=None*)
Exports the contents of the database.

Parameters

- **content_type** (*str*) – RDF content type. Defaults to 'text/turtle'
- **stream** (*bool*) – Chunk results? Defaults to False

- **chunk_size** (*int*) – Number of bytes to read per chunk when streaming. Defaults to 10240
- **graph_uri** (*str, optional*) – Named graph to export

Returns If stream = False

Return type str

Returns If stream = True

Return type gen

Examples

no streaming

```
>>> contents = conn.export()
```

streaming

```
>>> with conn.export(stream=True) as stream:
      contents = ''.join(stream)
```

graph(*query*, *content_type='text/turtle'*, ***kwargs*)

Executes a SPARQL graph query.

Parameters

- **query** (*str*) – SPARQL query
- **base_uri** (*str, optional*) – Base URI for the parsing of the query
- **limit** (*int, optional*) – Maximum number of results to return
- **offset** (*int, optional*) – Offset into the result set
- **timeout** (*int, optional*) – Number of ms after which the query should timeout. 0 or less implies no timeout
- **reasoning** (*bool, optional*) – Enable reasoning for the query
- **bindings** (*dict, optional*) – Map between query variables and their values
- **content_type** (*str*) – Content type for results. Defaults to 'text/turtle'

Returns Results in format given by content_type

Return type str

Examples

```
>>> conn.graph('construct {?s ?p ?o} where {?s ?p ?o}',
               offset=100, limit=100, reasoning=True)
```

bindings

```
>>> conn.graph('construct {?s ?p ?o} where {?s ?p ?o}',
               bindings={'o': '<urn:a>'})
```

graphql()

Makes a GraphQL object.

Returns A GraphQL object

Return type *GraphQL*

icv()
Makes an integrity constraint validation object.

Returns An ICV object

Return type *ICV*

is_consistent(*graph_uri=None*)
Checks if the database or named graph is consistent wrt its schema.

Parameters **graph_uri** (*str, optional*) – Named graph from which to check consistency

Returns Database consistency state

Return type bool

paths(*query*, *content_type='application/sparql-results+json'*, ***kwargs*)
Executes a SPARQL paths query.

Parameters

- **query** (*str*) – SPARQL query
- **base_uri** (*str, optional*) – Base URI for the parsing of the query
- **limit** (*int, optional*) – Maximum number of results to return
- **offset** (*int, optional*) – Offset into the result set
- **timeout** (*int, optional*) – Number of ms after which the query should timeout. 0 or less implies no timeout
- **reasoning** (*bool, optional*) – Enable reasoning for the query
- **bindings** (*dict, optional*) – Map between query variables and their values
- **content_type** (*str*) – Content type for results. Defaults to 'application/sparql-results+json'

Returns if content_type='application/sparql-results+json'.

Return type dict

Returns other content types.

Return type str

Examples

```
>>> conn.paths('paths start ?x = :subj end ?y = :obj via ?p',
               reasoning=True)
```

remove(*content*, *graph_uri=None*)
Removes data from the database.

Parameters

- **content** (Content) – Data to add
- **graph_uri** (*str, optional*) – Named graph from which to remove the data

Raises *stardog.exceptions.TransactionException* – If currently not in a transaction

Examples

```
>>> conn.remove(File('example.ttl'), graph_uri='urn:graph')
```

rollback()
Rolls back the current transaction.

Raises *stardog.exceptions.TransactionException* – If currently not in a transaction

select(*query*, *content_type='application/sparql-results+json'*, ***kwargs*)
Executes a SPARQL select query.

Parameters

- **query** (*str*) – SPARQL query
- **base_uri** (*str, optional*) – Base URI for the parsing of the query
- **limit** (*int, optional*) – Maximum number of results to return
- **offset** (*int, optional*) – Offset into the result set
- **timeout** (*int, optional*) – Number of ms after which the query should timeout. 0 or less implies no timeout
- **reasoning** (*bool, optional*) – Enable reasoning for the query
- **bindings** (*dict, optional*) – Map between query variables and their values
- **content_type** (*str, optional*) – Content type for results. Defaults to 'application/sparql-results+json'

Returns If content_type='application/sparql-results+json'

Return type dict

Returns Other content types

Return type str

Examples

```
>>> conn.select('select * {?s ?p ?o}',
                offset=100, limit=100, reasoning=True)
```

bindings

```
>>> conn.select('select * {?s ?p ?o}', bindings={'o': '<urn:a>'})
```

size(*exact=False*)
Database size.

Parameters **exact** (*bool, optional*) – Calculate the size exactly. Defaults to False

Returns The number of elements in database

Return type int

update(*query*, ***kwargs*)
Executes a SPARQL update query.

Parameters

- **query** (*str*) – SPARQL query
- **base_uri** (*str, optional*) – Base URI for the parsing of the query
- **limit** (*int, optional*) – Maximum number of results to return
- **offset** (*int, optional*) – Offset into the result set
- **timeout** (*int, optional*) – Number of ms after which the query should timeout. 0 or less implies no timeout
- **reasoning** (*bool, optional*) – Enable reasoning for the query
- **bindings** (*dict, optional*) – Map between query variables and their values

Examples

```
>>> conn.update('delete where {?s ?p ?o}')
```

class stardog.connection.**Docs**(*client*)
Bases: object

BITES: Document Storage.

See also:

https://www.stardog.com/docs/#_unstructured_data

__init__(*client*)
Initializes a Docs.

Use *stardog.connection.Connection.docs()* instead of constructing manually.

add(*name*, *content*)
Adds a document to the store.

Parameters

- **name** (*str*) – Name of the document
- **content** (Content) – Contents of the document

Examples

```
>>> docs.add('example', File('example.pdf'))
```

clear()
Removes all documents from the store.

delete(*name*)
Deletes a document from the store.

Parameters **name** (*str*) – Name of the document

get(*name*, *stream=False*, *chunk_size=10240*)
Gets a document from the store.

Parameters

- **name** (*str*) – Name of the document
- **stream** (*bool*) – If document should come in chunks or as a whole. Defaults to False
- **chunk_size** (*int*) – Number of bytes to read per chunk when streaming. Defaults to 10240

Returns If stream=False

Return type str

Returns If stream=True

Return type gen

Examples

no streaming

```
>>> contents = docs.get('example')
```

streaming

```
>>> with docs.get('example', stream=True) as stream:
                  contents = ''.join(stream)
```

size()
Calculates document store size.

Returns Number of documents in the store

Return type int

class stardog.connection.**GraphQL**(*conn*)
Bases: object

See also:

https://www.stardog.com/docs/#_graphql_queries

__init__(*conn*)
Initializes a GraphQL.

Use *stardog.connection.Connection.graphql()* instead of constructing manually.

add_schema(*name*, *content*)
Adds a schema to the database.

Parameters

- **name** (*str*) – Name of the schema
- **content** (Content) – Schema data

Examples

```
>>> gql.add_schema('people', content=File('people.graphql'))
```

clear_schemas()
Deletes all schemas.

query(*query*, *variables=None*)
Executes a GraphQL query.

Parameters

- **query** (*str*) – GraphQL query
- **variables** (*dict, optional*) – GraphQL variables. Keys: '@reasoning' (bool) to enable reasoning, '@schema' (str) to define schemas

Returns Query results

Return type dict

Examples

with schema and reasoning

```
>>> gql.query('{ Person {name} }',
              variables={'@reasoning': True, '@schema': 'people'})
```

with named variables

```
>>> gql.query(
      'query getPerson($id: Integer) { Person(id: $id) {name} }',
      variables={'id': 1000})
```

remove_schema(*name*)
Removes a schema from the database.

Parameters **name** (*str*) – Name of the schema

schema(*name*)
Gets schema information.

Parameters **name** (*str*) – Name of the schema

Returns GraphQL schema

Return type dict

schemas()
Retrieves all available schemas.

Returns All schemas

Return type dict

class stardog.connection.**ICV**(*conn*)
Bases: object

Integrity Constraint Validation.

See also:

https://www.stardog.com/docs/#_validating_constraints

__init__(*conn*)
Initializes an ICV.

Use *stardog.connection.Connection.icv()* instead of constructing manually.

add(*content*)
Adds integrity constraints to the database.

Parameters **content** (Content) – Data to add

Examples

```
>>> icv.add(File('constraints.ttl'))
```

clear()
Removes all integrity constraints from the database.

convert(*content*, *graph_uri=None*)
Converts given integrity constraints to a SPARQL query.

Parameters

- **content** (Content) – Integrity constraints
- **graph_uri** (*str, optional*) – Named graph from which to apply constraints

Returns SPARQL query

Return type str

Examples

```
>>> icv.convert(File('constraints.ttl'), graph_uri='urn:graph')
```

explain_violations(*content*, *graph_uri=None*)
Explains violations of the given integrity constraints.

Parameters

- **content** (Content) – Data to check for violations
- **graph_uri** (*str, optional*) – Named graph from which to check for validations

Returns Integrity constraint violations

Return type dict

Examples

```
>>> icv.explain_violations(File('constraints.ttl'),
                           graph_uri='urn:graph')
```

is_valid(*content*, *graph_uri=None*)
Checks if given integrity constraints are valid.

Parameters

- **content** (Content) – Data to check for validity
- **graph_uri** (*str, optional*) – Named graph to check for validity

Returns Integrity constraint validity

Return type bool

Examples

```
>>> icv.is_valid(File('constraints.ttl'), graph_uri='urn:graph')
```

remove(*content*)
Removes integrity constraints from the database.

Parameters **content** (Content) – Data to remove

Examples

```
>>> icv.remove(File('constraints.ttl'))
```

2.2 stardog.admin

Administer a Stardog server.

class stardog.admin.**Admin**(*endpoint=None*, *username=None*, *password=None*, *auth=None*)
Bases: object

Admin Connection.

This is the entry point for admin-related operations on a Stardog server.

See also:

https://www.stardog.com/docs/#_administering_stardog

__init__(*endpoint=None*, *username=None*, *password=None*, *auth=None*)
Initializes an admin connection to a Stardog server.

Parameters

- **endpoint** (*str, optional*) – Url of the server endpoint. Defaults to *http://localhost:5820*
- **username** (*str, optional*) – Username to use in the connection. Defaults to *admin*
- **password** (*str, optional*) – Password to use in the connection. Defaults to *admin*

auth (requests.auth.AuthBase, optional): requests Authentication object. Defaults to *None*

auth and username/password should not be used together. If the are the value of *auth* will take precedent.
.. rubric:: Examples

```
>>> admin = Admin(endpoint='http://localhost:9999',
                  username='admin', password='admin')
```

clear_stored_queries()
Remove all stored queries on the server.

database(*name*)
Retrieves an object representing a database.

Parameters **name** (*str*) – The database name

Returns The requested database

Return type *Database*

databases()
Retrieves all databases.

Returns A list of database objects

Return type list[*Database*]

kill_query(*id*)
Kills a running query.

Parameters **id** (*str*) – ID of the query to kill

new_database(*name*, *options=None*, **contents*)
Creates a new database.

Parameters

- **name** (*str*) – the database name
- **options** (*dict*) – Dictionary with database options (optional)
- ***contents** (*Content or (Content, str), optional*) – Datasets to perform bulk-load with. Named graphs are made with tuples of Content and the name.

Returns The database object

Return type *Database*

Examples

Options

```
>>> admin.new_database('db', {'search.enabled': True})
```

bulk-load

```
>>> admin.new_database('db', {},
                       File('example.ttl'), File('test.rdf'))
```

bulk-load to named graph

```
>>> admin.new_database('db', {}, (File('test.rdf'), 'urn:context'))
```

new_role(*name*)
Creates a new role.

Parameters **name** (*str*) – The name of the new Role

Returns The new Role object

Return type *Role*

new_stored_query(*name*, *query*, *options=None*)
Creates a new Stored Query.

Parameters

- **name** (*str*) – The name of the stored query
- **query** (*str*) – The query text
- **options** (*dict, optional*) – Additional options

Returns the new StoredQuery

Return type *StoredQuery*

Examples

```
>>> admin.new_stored_query('all triples',
      'select * where { ?s ?p ?o . }',
      { 'database': 'mydb' }
    )
```

new_user(*username*, *password*, *superuser=False*)
Creates a new user.

Parameters

- **username** (*str*) – The username
- **password** (*str*) – The password
- **superuser** (*bool*) – Should the user be super? Defaults to false.

Returns The new User object

Return type *User*

new_virtual_graph(*name*, *mappings*, *options*)
Creates a new Virtual Graph.

Parameters

- **name** (*str*) – The name of the virtual graph
- **mappings** (Content) – New mapping contents
- **options** (*dict*) – Options for the new virtual graph

Returns the new VirtualGraph

Return type *VirtualGraph*

Examples

```
>>> admin.new_virtual_graph(
      'users', File('mappings.ttl'),
      {'jdbc.driver': 'com.mysql.jdbc.Driver'}
    )
```

queries()
Gets information about all running queries.

Returns Query information

Return type dict

query(*id*)
Gets information about a running query.

Parameters **id** (*str*) – Query ID

Returns Query information

Return type dict

restore (*from_path*, ***, *name=None*, *force=False*)
Restore a database.

Parameters

- **from_path** (*str*) – the full path on the server to the backup
- **name** (*str, optional*) – the name of the database to restore to if different from the backup
- **force** (*boolean, optional*) – a backup will not be restored over an existing database of the same name; the force flag should be used to overwrite the database

Examples

```
>>> admin.restore("/data/stardog/.backup/db/2019-12-01")
>>> admin.restore("/data/stardog/.backup/db/2019-11-05",
                  name="db2", force=True)
```

See also:

https://www.stardog.com/docs/#_restore_a_database_from_a_backup

role (*name*)
Retrieves an object representing a role.

Parameters **name** (*str*) – The name of the Role

Returns The Role object

Return type *Role*

roles ()
Retrieves all roles.

Returns A list of Role objects

Return type list[*Role*]

shutdown ()
Shuts down the server.

stored_queries ()
Retrieves all stored queries.

Returns A list of StoredQuery objects

Return type list[*StoredQuery*]

stored_query (*name*)
Retrieves a Stored Query.

Parameters **name** (*str*) – The name of the Stored Query to retrieve

Returns The StoredQuery object

Return type *StoredQuery*

user (*name*)
Retrieves an object representing a user.

Parameters **name** (*str*) – The name of the user

Returns The User object

Return type *User*

users()
Retrieves all users.

Returns A list of User objects

Return type list[*User*]

validate()
Validates an admin connection.

Returns The connection state

Return type bool

virtual_graph(*name*)
Retrieves a Virtual Graph.

Parameters **name** (*str*) – The name of the Virtual Graph to retrieve

Returns The VirtualGraph object

Return type *VirtualGraph*

virtual_graphs()
Retrieves all virtual graphs.

Returns A list of VirtualGraph objects

Return type list[*VirtualGraph*]

class stardog.admin.**Database**(*name*, *client*)
Bases: object

Database Admin

See also:

https://www.stardog.com/docs/#_database_admin

__init__(*name*, *client*)
Initializes a Database.

Use *stardog.admin.Admin.database()*, *stardog.admin.Admin.databases()*, or *stardog.admin.Admin.new_database()* instead of constructing manually.

backup(*, *to=None*)
Backup a database.

Parameters **to** (*string, optional*) – specify a path on the server to store the backup

See also:

https://www.stardog.com/docs/#_backup_a_database

copy(*to*)
Makes a copy of this database under another name.

The database must be offline.

Parameters **to** (*str*) – Name of the new database to be created

Returns The new Database

Return type *Database*

drop()
Drops the database.

get_options(**options*)
Gets database options.

Parameters ***options** (*str*) – Database option names

Returns Database options

Return type dict

Examples

```
>>> db.get_options('search.enabled', 'spatial.enabled')
```

property name
The name of the database.

offline()
Sets a database to offline state.

online()
Sets a database to online state.

optimize()
Optimizes a database.

repair()
Repairs a database.

The database must be offline.

set_options(*options*)
Sets database options.

The database must be offline.

Parameters **options** (*dict*) – Database options

Examples

```
>>> db.set_options({'spatial.enabled': False})
```

class stardog.admin.**Role**(*name*, *client*)
Bases: object

See also:

https://www.stardog.com/docs/#_security

__init__(*name*, *client*)
Initializes a Role.

Use *stardog.admin.Admin.role()*, *stardog.admin.Admin.roles()*, or *stardog.admin.Admin.new_role()* instead of constructing manually.

add_permission(*action*, *resource_type*, *resource*)
Adds a permission to the role.

See also:

https://www.stardog.com/docs/#_permissions

Parameters

- **action** (*str*) – Action type (e.g., 'read', 'write')
- **resource_type** (*str*) – Resource type (e.g., 'user', 'db')
- **resource** (*str*) – Target resource (e.g., 'username', '*')

Examples

```
>>> role.add_permission('read', 'user', 'username')
>>> role.add_permission('write', '*', '*')
```

delete (*force=None*)
Deletes the role.

Parameters **force** (*bool*) – Force deletion of the role

property name
The name of the Role.

permissions ()
Gets the role permissions.

See also:

https://www.stardog.com/docs/#_permissions

Returns Role permissions

Return type dict

remove_permission (*action*, *resource_type*, *resource*)
Removes a permission from the role.

See also:

https://www.stardog.com/docs/#_permissions

Parameters

- **action** (*str*) – Action type (e.g., 'read', 'write')
- **resource_type** (*str*) – Resource type (e.g., 'user', 'db')
- **resource** (*str*) – Target resource (e.g., 'username', '*')

Examples

```
>>> role.remove_permission('read', 'user', 'username')
>>> role.remove_permission('write', '*', '*')
```

users ()
Lists the users for this role.

Returns list[User]

class stardog.admin.**StoredQuery**(*name*, *client*)
Bases: object

Stored Query

See also:

https://www.stardog.com/docs/#_list_stored_queries https://www.stardog.com/docs/#_managing_stored_queries

__init__(*name*, *client*)
Initializes a stored query.

Use *stardog.admin.Admin.stored_query()*, *stardog.admin.Admin.stored_queries()*, or *stardog.admin.Admin.new_stored_query()* instead of constructing manually.

property creator
The creator of the stored query.

property database
The database the stored query applies to.

delete()
Deletes the Stored Query.

property description
The description of the stored query.

property name
The name of the stored query.

property query
The text of the stored query.

property reasoning
The value of the reasoning property.

property shared
The value of the shared property.

update(***options*)
Updates the Stored Query.

Parameters ****options** (*str*) – Named arguments to update.

Examples

Update description

```
>>> stored_query.update(description='this query finds all the relevant...')
```

class stardog.admin.**User**(*name*, *client*)
Bases: object

See also:

https://www.stardog.com/docs/#_security

__init__(*name*, *client*)
Initializes a User.

Use *stardog.admin.Admin.user()*, *stardog.admin.Admin.users()*, or *stardog.admin.Admin.new_user()* instead of constructing manually.

add_permission(*action*, *resource_type*, *resource*)
Add a permission to the user.

See also:

https://www.stardog.com/docs/#_permissions

Parameters

- **action** (*str*) – Action type (e.g., 'read', 'write')
- **resource_type** (*str*) – Resource type (e.g., 'user', 'db')
- **resource** (*str*) – Target resource (e.g., 'username', '*')

Examples

```
>>> user.add_permission('read', 'user', 'username')
>>> user.add_permission('write', '*', '*')
```

add_role(*role*)
Adds an existing role to the user.

Parameters **role** (*str or Role*) – The role to add or its name

Examples

```
>>> user.add_role('reader')
>>> user.add_role(admin.role('reader'))
```

delete()
Deletes the user.

effective_permissions()
Gets the user's effective permissions.

Returns User effective permissions

Return type dict

is_enabled()
Checks if the user is enabled.

Returns User activation state

Return type bool

is_superuser()
Checks if the user is a super user.

Returns Superuser state

Return type bool

property name
The user name.

Type str

permissions()
Gets the user permissions.

See also:

https://www.stardog.com/docs/#_permissions

Returns User permissions

Return type dict

remove_permission(*action*, *resource_type*, *resource*)
Removes a permission from the user.

See also:

https://www.stardog.com/docs/#_permissions

Parameters

- **action** (*str*) – Action type (e.g., 'read', 'write')
- **resource_type** (*str*) – Resource type (e.g., 'user', 'db')
- **resource** (*str*) – Target resource (e.g., 'username', '*')

Examples

```
>>> user.remove_permission('read', 'user', 'username')
>>> user.remove_permission('write', '*', '*')
```

remove_role(*role*)
Removes a role from the user.

Parameters **role** (*str or Role*) – The role to remove or its name

Examples

```
>>> user.remove_role('reader')
>>> user.remove_role(admin.role('reader'))
```

roles()
Gets all the User's roles.

Returns list[Role]

set_enabled(*enabled*)
Enables or disables the user.

Parameters **enabled** (*bool*) – Desired User state

set_password(*password*)
Sets a new password.

Parameters **password** (*str*) –

set_roles(**roles*)
Sets the roles of the user.

Parameters ***roles** (*str or Role*) – The roles to add the User to

Examples

```
>>> user.set_roles('reader', admin.role('writer'))
```

class stardog.admin.**VirtualGraph**(*name*, *client*)
Bases: object

Virtual Graph

See also:

https://www.stardog.com/docs/#_structured_data

__init__(*name*, *client*)
Initializes a virtual graph.

Use *stardog.admin.Admin.virtual_graph()*, *stardog.admin.Admin.virtual_graphs()*, or *stardog.admin.Admin.new_virtual_graph()* instead of constructing manually.

available()
Checks if the Virtual Graph is available.

Returns Availability state

Return type bool

delete()
Deletes the Virtual Graph.

mappings(*content_type='text/turtle'*)
Gets the Virtual Graph mappings.

Parameters **content_type** (*str*) – Content type for results. Defaults to 'text/turtle'

Returns Mappings in given content type

Return type str

property name
The name of the virtual graph.

options()
Gets Virtual Graph options.

Returns Options

Return type dict

update(*name*, *mappings*, *options*)
Updates the Virtual Graph.

Parameters

- **name** (*str*) – The new name
- **mappings** (Content) – New mapping contents
- **options** (*dict*) – New options

Examples

```
>>> vg.update('users', File('mappings.ttl'),
              {'jdbc.driver': 'com.mysql.jdbc.Driver'})
```

2.3 stardog.content

Content that can be loaded into Stardog.

class stardog.content.**Content**
Bases: object

Content base class.

class stardog.content.**File**(*fname*, *content_type=None*, *content_encoding=None*, *name=None*)
Bases: *stardog.content.Content*

File-based content.

__init__(*fname*, *content_type=None*, *content_encoding=None*, *name=None*)
Initializes a File object.

Parameters

- **fname** (*str*) – Filename
- **content_type** (*str, optional*) – Content type. It will be automatically detected from the filename
- **content_encoding** (*str, optional*) – Content encoding. It will be automatically detected from the filename
- **name** (*str, optional*) – Object name. It will be automatically detected from the filename

Examples

```
>>> File('data.ttl')
>>> File('data.doc', 'application/msword')
```

data()

class stardog.content.**Raw**(*content*, *content_type=None*, *content_encoding=None*, *name=None*)
Bases: *stardog.content.Content*

User-defined content.

__init__(*content*, *content_type=None*, *content_encoding=None*, *name=None*)
Initializes a Raw object.

Parameters

- **content** (*obj*) – Object representing the content (e.g., str, file)
- **content_type** (*str, optional*) – Content type
- **content_encoding** (*str, optional*) – Content encoding
- **name** (*str, optional*) – Object name

Examples

```
>>> Raw(':luke a :Human', 'text/turtle', name='data.ttl')
>>> Raw(open('data.ttl.zip', 'rb'),
        'text/turtle', 'zip', 'data.ttl')
```

data()

class stardog.content.**URL**(*url*, *content_type=None*, *content_encoding=None*, *name=None*)
Bases: *stardog.content.Content*

Url-based content.

__init__(*url*, *content_type=None*, *content_encoding=None*, *name=None*)
Initializes a URL object.

Parameters

- **url** (*str*) – Url
- **content_type** (*str, optional*) – Content type. It will be automatically detected from the url
- **content_encoding** (*str, optional*) – Content encoding. It will be automatically detected from the filename
- **name** (*str, optional*) – Object name. It will be automatically detected from the url

Examples

```
>>> URL('http://example.com/data.ttl')
>>> URL('http://example.com/data.doc', 'application/msword')
```

data()

2.4 stardog.exceptions

exception stardog.exceptions.**StardogException**
Bases: Exception

General Stardog Exceptions

exception stardog.exceptions.**TransactionException**
Bases: *stardog.exceptions.StardogException*

Transaction Exceptions

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

E

F

G

I

K

M

N

O

P

Q

R

S

T

U

V