

---

# **pystardog Documentation**

**Pedro Oliveira, John Bresnahan, Stephen Nowell**

**May 25, 2021**



**CONTENTS:**

<b>1</b>	<b>pystardog</b>	<b>1</b>
1.1	What is it? . . . . .	1
1.2	Installation . . . . .	1
1.3	Documentation . . . . .	1
1.4	Tests . . . . .	1
1.5	Quick Example . . . . .	2
1.6	Interactive Tutorial . . . . .	2
<b>2</b>	<b>Modules</b>	<b>3</b>
2.1	stardog.connection . . . . .	3
2.2	stardog.admin . . . . .	13
2.3	stardog.content . . . . .	26
2.4	stardog.exceptions . . . . .	27
<b>3</b>	<b>Indices and tables</b>	<b>29</b>
	<b>Python Module Index</b>	<b>31</b>
	<b>Index</b>	<b>33</b>



## **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**

To run the tests locally, a valid Stardog license is required and placed in the `dockerfiles/stardog-license-key.bin`. Docker and docker-compose are also required.

```
docker-compose up --exit-code-from tests
```

## 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.

## 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](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](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](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')
```

**list()**

List all integrity constraints from the database.

**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](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.

**cluster\_info()**

Prints info about the nodes in the Stardog Pack cluster.

**Returns** Nodes of the cluster.

**Return type** dict

**cluster\_status** ()

Prints status information for each node in the cluster

**Returns** Nodes of the cluster and extra information

**Return type** dict

**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*]

**datasource** (*name*)

Retrieves an object representing a DataSource.

**Parameters** **name** (*str*) – The name of the data source

**Returns** The DataSource object

**Return type** *DataSource*

**datasources** ()

Retrieves all data sources.

**Returns** A list of DataSources

**Return type** list[DataSources]

**datasources\_info** ()

List data sources info

**Returns** A list of data sources info

**Return type** list

**get\_server\_properties** ()

Get the value of any set server-level properties

**Returns** dict: Server properties

**import\_virtual\_graph** (*db, mappings, named\_graph, remove\_all, options*)

Import (materialize) a virtual graph directly into the local knowledge graph.

**Parameters**

- **db** (*str*) – The database into which to import the graph
- **mappings** (*Content*) – New mapping contents. An empty string can be passed for autogenerated mappings.
- **named\_graph** (*str*) – Name of the graph into which import the VG.
- **remove\_all** (*bool*) – Should the target named graph be cleared before importing?

- **options** (*dict*) – Options for the new virtual graph, <https://docs.stardog.com/virtual-graphs/virtual-graph-configuration#virtual-graph-properties>

## Examples

```
>>> admin.import_virtual_graph(
    'db-name', mappings=File('mappings.ttl'),
    named_graph='my-graph', remove_all=True, options={'jdbc.driver': 'com.
mysql.jdbc.Driver'})
```

### 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\_datasource(*name*, *options*)

Creates a new DataSource.

#### Parameters

- **name** (*str*) – The name of the data source
- **options** (*dict*) – Data source options

**Returns** The new DataSource object

**Return type** *User*

**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={}, datasource=None, db=None*)

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](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.**DataSource** (*name, client*)

Bases: object

Initializes a DataSource

**See also:**

<https://docs.stardog.com/virtual-graphs/virtual-sources>

**\_\_init\_\_** (*name, client*)

Initializes a DataSource.

Use `stardog.admin.Admin.data_source()`, `stardog.admin.Admin.data_sources()`, or `stardog.admin.Admin.new_data_source()` instead of constructing manually.

**available** ()

Checks if the data source is available.

**Returns** Availability state

**Return type** bool

**delete** ()

Deletes a data source

**get\_options** ()

Get data source options

**info** ()

Get data source info

**Returns** Info

**Return type** dict

**property name**

The name of the data source.

**online** ()

Online a data source

**refresh\_count** (*meta=None*)

Refresh table row-count estimates

**refresh\_metadata** (*meta=None*)

Refresh metadata

**share** ()

Share data source

**update** (*options=None*)

Update data source

**class** `stardog.admin.Database` (*name, client*)

Bases: object

Database Admin

**See also:**

[https://www.stardog.com/docs/#\\_database\\_admin](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](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](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](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](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](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/#_list_stored_queries)

[https://www.stardog.com/docs/#\\_managing\\_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](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](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](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](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](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.

**get\_database** ()

Gets database associated with the VirtualGraph.

**Returns** Database name

**Return type** string

**info** ()

Gets Virtual Graph info.

**Returns** Info

**Return type** dict

**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*={}, *datasource*=None, *db*=None)

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



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### S

`stardog.admin`, [13](#)  
`stardog.connection`, [3](#)  
`stardog.content`, [26](#)  
`stardog.exceptions`, [27](#)



## Symbols

`__init__()` (*stardog.admin.Admin* method), 13  
`__init__()` (*stardog.admin.DataSource* method), 18  
`__init__()` (*stardog.admin.Database* method), 19  
`__init__()` (*stardog.admin.Role* method), 21  
`__init__()` (*stardog.admin.StoredQuery* method), 22  
`__init__()` (*stardog.admin.User* method), 23  
`__init__()` (*stardog.admin.VirtualGraph* method), 25  
`__init__()` (*stardog.connection.Connection* method), 3  
`__init__()` (*stardog.connection.Docs* method), 9  
`__init__()` (*stardog.connection.GraphQL* method), 10  
`__init__()` (*stardog.connection.ICV* method), 11  
`__init__()` (*stardog.content.File* method), 26  
`__init__()` (*stardog.content.Raw* method), 27  
`__init__()` (*stardog.content.URL* method), 27

## A

`add()` (*stardog.connection.Connection* method), 3  
`add()` (*stardog.connection.Docs* method), 9  
`add()` (*stardog.connection.ICV* method), 11  
`add_permission()` (*stardog.admin.Role* method), 21  
`add_permission()` (*stardog.admin.User* method), 23  
`add_role()` (*stardog.admin.User* method), 23  
`add_schema()` (*stardog.connection.GraphQL* method), 10  
`Admin` (class in *stardog.admin*), 13  
`ask()` (*stardog.connection.Connection* method), 4  
`available()` (*stardog.admin.DataSource* method), 19  
`available()` (*stardog.admin.VirtualGraph* method), 25

## B

`backup()` (*stardog.admin.Database* method), 19  
`begin()` (*stardog.connection.Connection* method), 4

## C

`clear()` (*stardog.connection.Connection* method), 4  
`clear()` (*stardog.connection.Docs* method), 9  
`clear()` (*stardog.connection.ICV* method), 12

`clear_schemas()` (*stardog.connection.GraphQL* method), 10  
`clear_stored_queries()` (*stardog.admin.Admin* method), 13  
`close()` (*stardog.connection.Connection* method), 5  
`cluster_info()` (*stardog.admin.Admin* method), 13  
`cluster_status()` (*stardog.admin.Admin* method), 14  
`commit()` (*stardog.connection.Connection* method), 5  
`Connection` (class in *stardog.connection*), 3  
`Content` (class in *stardog.content*), 26  
`convert()` (*stardog.connection.ICV* method), 12  
`copy()` (*stardog.admin.Database* method), 20  
`creator()` (*stardog.admin.StoredQuery* property), 22

## D

`data()` (*stardog.content.File* method), 26  
`data()` (*stardog.content.Raw* method), 27  
`data()` (*stardog.content.URL* method), 27  
`Database` (class in *stardog.admin*), 19  
`database()` (*stardog.admin.Admin* method), 14  
`database()` (*stardog.admin.StoredQuery* property), 22  
`databases()` (*stardog.admin.Admin* method), 14  
`DataSource` (class in *stardog.admin*), 18  
`datasource()` (*stardog.admin.Admin* method), 14  
`datasources()` (*stardog.admin.Admin* method), 14  
`datasources_info()` (*stardog.admin.Admin* method), 14  
`delete()` (*stardog.admin.DataSource* method), 19  
`delete()` (*stardog.admin.Role* method), 21  
`delete()` (*stardog.admin.StoredQuery* method), 22  
`delete()` (*stardog.admin.User* method), 23  
`delete()` (*stardog.admin.VirtualGraph* method), 25  
`delete()` (*stardog.connection.Docs* method), 9  
`description()` (*stardog.admin.StoredQuery* property), 22  
`Docs` (class in *stardog.connection*), 9  
`docs()` (*stardog.connection.Connection* method), 5  
`drop()` (*stardog.admin.Database* method), 20

## E

`effective_permissions()` (*stardog.admin.User*

[method](#)), 23  
[explain\(\)](#) (*stardog.connection.Connection method*), 5  
[explain\\_inconsistency\(\)](#) (*stardog.connection.Connection method*), 5  
[explain\\_inference\(\)](#) (*stardog.connection.Connection method*), 5  
[explain\\_violations\(\)](#) (*stardog.connection.ICV method*), 12  
[export\(\)](#) (*stardog.connection.Connection method*), 5

## F

[File](#) (*class in stardog.content*), 26

## G

[get\(\)](#) (*stardog.connection.Docs method*), 9  
[get\\_database\(\)](#) (*stardog.admin.VirtualGraph method*), 25  
[get\\_options\(\)](#) (*stardog.admin.Database method*), 20  
[get\\_options\(\)](#) (*stardog.admin.DataSource method*), 19  
[get\\_server\\_properties\(\)](#) (*stardog.admin.Admin method*), 14  
[graph\(\)](#) (*stardog.connection.Connection method*), 6  
[GraphQL](#) (*class in stardog.connection*), 10  
[graphql\(\)](#) (*stardog.connection.Connection method*), 6

## I

[ICV](#) (*class in stardog.connection*), 11  
[icv\(\)](#) (*stardog.connection.Connection method*), 7  
[import\\_virtual\\_graph\(\)](#) (*stardog.admin.Admin method*), 14  
[info\(\)](#) (*stardog.admin.DataSource method*), 19  
[info\(\)](#) (*stardog.admin.VirtualGraph method*), 25  
[is\\_consistent\(\)](#) (*stardog.connection.Connection method*), 7  
[is\\_enabled\(\)](#) (*stardog.admin.User method*), 23  
[is\\_superuser\(\)](#) (*stardog.admin.User method*), 23  
[is\\_valid\(\)](#) (*stardog.connection.ICV method*), 12

## K

[kill\\_query\(\)](#) (*stardog.admin.Admin method*), 15

## L

[list\(\)](#) (*stardog.connection.ICV method*), 13

## M

[mappings\(\)](#) (*stardog.admin.VirtualGraph method*), 25

## N

[name\(\)](#) (*stardog.admin.Database property*), 20  
[name\(\)](#) (*stardog.admin.DataSource property*), 19  
[name\(\)](#) (*stardog.admin.Role property*), 21

[name\(\)](#) (*stardog.admin.StoredQuery property*), 22  
[name\(\)](#) (*stardog.admin.User property*), 24  
[name\(\)](#) (*stardog.admin.VirtualGraph property*), 25  
[new\\_database\(\)](#) (*stardog.admin.Admin method*), 15  
[new\\_datasource\(\)](#) (*stardog.admin.Admin method*), 15  
[new\\_role\(\)](#) (*stardog.admin.Admin method*), 15  
[new\\_stored\\_query\(\)](#) (*stardog.admin.Admin method*), 16  
[new\\_user\(\)](#) (*stardog.admin.Admin method*), 16  
[new\\_virtual\\_graph\(\)](#) (*stardog.admin.Admin method*), 16

## O

[offline\(\)](#) (*stardog.admin.Database method*), 20  
[online\(\)](#) (*stardog.admin.Database method*), 20  
[online\(\)](#) (*stardog.admin.DataSource method*), 19  
[optimize\(\)](#) (*stardog.admin.Database method*), 20  
[options\(\)](#) (*stardog.admin.VirtualGraph method*), 25

## P

[paths\(\)](#) (*stardog.connection.Connection method*), 7  
[permissions\(\)](#) (*stardog.admin.Role method*), 21  
[permissions\(\)](#) (*stardog.admin.User method*), 24

## Q

[queries\(\)](#) (*stardog.admin.Admin method*), 17  
[query\(\)](#) (*stardog.admin.Admin method*), 17  
[query\(\)](#) (*stardog.admin.StoredQuery property*), 22  
[query\(\)](#) (*stardog.connection.GraphQL method*), 10

## R

[Raw](#) (*class in stardog.content*), 26  
[reasoning\(\)](#) (*stardog.admin.StoredQuery property*), 22  
[refresh\\_count\(\)](#) (*stardog.admin.DataSource method*), 19  
[refresh\\_metadata\(\)](#) (*stardog.admin.DataSource method*), 19  
[remove\(\)](#) (*stardog.connection.Connection method*), 7  
[remove\(\)](#) (*stardog.connection.ICV method*), 13  
[remove\\_permission\(\)](#) (*stardog.admin.Role method*), 21  
[remove\\_permission\(\)](#) (*stardog.admin.User method*), 24  
[remove\\_role\(\)](#) (*stardog.admin.User method*), 24  
[remove\\_schema\(\)](#) (*stardog.connection.GraphQL method*), 11  
[repair\(\)](#) (*stardog.admin.Database method*), 20  
[restore\(\)](#) (*stardog.admin.Admin method*), 17  
[Role](#) (*class in stardog.admin*), 20  
[role\(\)](#) (*stardog.admin.Admin method*), 17  
[roles\(\)](#) (*stardog.admin.Admin method*), 17

`roles()` (*stardog.admin.User method*), 24  
`rollback()` (*stardog.connection.Connection method*),  
 8

## S

`schema()` (*stardog.connection.GraphQL method*), 11  
`schemas()` (*stardog.connection.GraphQL method*), 11  
`select()` (*stardog.connection.Connection method*), 8  
`set_enabled()` (*stardog.admin.User method*), 24  
`set_options()` (*stardog.admin.Database method*),  
 20  
`set_password()` (*stardog.admin.User method*), 24  
`set_roles()` (*stardog.admin.User method*), 25  
`share()` (*stardog.admin.DataSource method*), 19  
`shared()` (*stardog.admin.StoredQuery property*), 22  
`shutdown()` (*stardog.admin.Admin method*), 17  
`size()` (*stardog.connection.Connection method*), 8  
`size()` (*stardog.connection.Docs method*), 10  
`stardog.admin` (*module*), 13  
`stardog.connection` (*module*), 3  
`stardog.content` (*module*), 26  
`stardog.exceptions` (*module*), 27  
`StardogException`, 27  
`stored_queries()` (*stardog.admin.Admin method*),  
 18  
`stored_query()` (*stardog.admin.Admin method*), 18  
`StoredQuery` (*class in stardog.admin*), 22

## T

`TransactionException`, 28

## U

`update()` (*stardog.admin.DataSource method*), 19  
`update()` (*stardog.admin.StoredQuery method*), 22  
`update()` (*stardog.admin.VirtualGraph method*), 26  
`update()` (*stardog.connection.Connection method*), 8  
`URL` (*class in stardog.content*), 27  
`User` (*class in stardog.admin*), 22  
`user()` (*stardog.admin.Admin method*), 18  
`users()` (*stardog.admin.Admin method*), 18  
`users()` (*stardog.admin.Role method*), 22

## V

`validate()` (*stardog.admin.Admin method*), 18  
`virtual_graph()` (*stardog.admin.Admin method*),  
 18  
`virtual_graphs()` (*stardog.admin.Admin method*),  
 18  
`VirtualGraph` (*class in stardog.admin*), 25