SPARQLWrapper Documentation

CHANGE_AUTHORS

Dec 22, 2019

Basic steps

SPARQLWrapper is a simple Python wrapper around a SPARQL service to remotelly execute your queries. It helps in creating the query invokation and, possibly, convert the result into a more manageable format.

CHAPTER 1

SPARQL Endpoint interface to Python

1.1 About

SPARQLWrapper is a simple Python wrapper around a SPARQL service to remotelly execute your queries. It helps in creating the query invokation and, possibly, convert the result into a more manageable format.

1.2 Installation & Distribution

You can install SPARQLWrapper from PyPi:

```
$ pip install sparqlwrapper
```

You can install SPARQLWrapper from GitHub:

```
$ pip install git+https://github.com/rdflib/sparqlwrapper#egg=sparqlwrapper
```

You can install SPARQLWrapper from Debian:

```
$ sudo apt-get install python-sparqlwrapper
```

Note: Be aware that there could be a gap between the latest version of SPARQLWrapper and the version available as Debian package.

Also, the source code of the package can be downloaded in `.zip` and `.tar.gz` formats from GitHub SPARQLWrapper releases. Documentation is included in the distribution.

1.3 How to use

1.3.1 First steps

The simplest usage of this module looks as follows (using the default, ie, XML return format, and special URI for the SPARQL Service):

```
from SPARQLWrapper import SPARQLWrapper

queryString = "SELECT * WHERE { ?s ?p ?o. }"
sparql = SPARQLWrapper("http://example.org/sparql")

sparql.setQuery(queryString)

try :
   ret = sparql.query()
   # ret is a stream with the results in XML, see <http://www.w3.org/TR/rdf-sparql-
↪XMLres/>
except :
   deal_with_the_exception()
```

If `SPARQLWrapper("http://example.org/sparql",returnFormat=SPARQLWrapper.JSON)` was used, the result would be in JSON format instead of XML.

1.3.2 SELECT example

```
from SPARQLWrapper import SPARQLWrapper, JSON

sparql = SPARQLWrapper("http://dbpedia.org/sparql")
sparql.setQuery("""
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    SELECT ?label
    WHERE { <http://dbpedia.org/resource/Asturias> rdfs:label ?label }
""")
sparql.setReturnFormat(JSON)
results = sparql.query().convert()

for result in results["results"]["bindings"]:
    print(result["label"]["value"])

print('----------------------------')

for result in results["results"]["bindings"]:
    print('%s: %s' % (result["label"]["xml:lang"], result["label"]["value"]))
```

1.3.3 ASK example

```
from SPARQLWrapper import SPARQLWrapper, XML

sparql = SPARQLWrapper("http://dbpedia.org/sparql")
sparql.setQuery("""
    ASK WHERE {
        <http://dbpedia.org/resource/Asturias> rdfs:label "Asturias"@es
```

(continues on next page)

(continued from previous page)

```
    }
""")
sparql.setReturnFormat(XML)
results = sparql.query().convert()
print(results.toxml())
```

1.3.4 CONSTRUCT example

```
from SPARQLWrapper import SPARQLWrapper, RDFXML
from rdflib import Graph

sparql = SPARQLWrapper("http://dbpedia.org/sparql")

sparql.setQuery("""
    PREFIX dbo: <http://dbpedia.org/ontology/>
    PREFIX schema: <http://schema.org/>

    CONSTRUCT {
      ?lang a schema:Language ;
      schema:alternateName ?iso6391Code .
    }
    WHERE {
      ?lang a dbo:Language ;
      dbo:iso6391Code ?iso6391Code .
      FILTER (STRLEN(?iso6391Code)=2) # to filter out non-valid values
    }
""")

sparql.setReturnFormat(RDFXML)
results = sparql.query().convert()
print(results.serialize(format='xml'))
```

1.3.5 DESCRIBE example

```
from SPARQLWrapper import SPARQLWrapper, N3
from rdflib import Graph

sparql = SPARQLWrapper("http://dbpedia.org/sparql")

sparql.setQuery("""
    DESCRIBE <http://dbpedia.org/resource/Asturias>
""")

sparql.setReturnFormat(N3)
results = sparql.query().convert()
g = Graph()
g.parse(data=results, format="n3")
print(g.serialize(format='n3'))
```

1.3.6 SPARQL UPDATE example

```
from SPARQLWrapper import SPARQLWrapper, POST, DIGEST

sparql = SPARQLWrapper("https://example.org/sparql-auth")

sparql.setHTTPAuth(DIGEST)
sparql.setCredentials("login", "password")
sparql.setMethod(POST)

sparql.setQuery("""
WITH <http://example.graph>
DELETE
{ <http://dbpedia.org/resource/Asturias> rdfs:label "Asturies"@ast }
""")

results = sparql.query()
print results.response.read()
```

1.3.7 SPARQLWrapper2 example

There is also a `SPARQLWrapper2` class that works with JSON SELECT results only and wraps the results to make processing of average queries a bit simpler.

```
from SPARQLWrapper import SPARQLWrapper2

sparql = SPARQLWrapper2("http://dbpedia.org/sparql")
sparql.setQuery("""
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    SELECT ?label
    WHERE { <http://dbpedia.org/resource/Asturias> rdfs:label ?label }
""")

for result in sparql.query().bindings:
    print('%s: %s' % (result["label"].lang, result["label"].value))
```

1.3.8 Return formats

The expected return formats differs from the query type (`SELECT`, `ASK`, `CONSTRUCT`, `DESCRIBE`...).

Note: From the SPARQL specification, *The response body of a successful query operation with a 2XX response is either:*

- `SELECT` and `ASK`: a SPARQL Results Document in XML, JSON, or CSV/TSV format.
- `DESCRIBE` and `CONSTRUCT`: an RDF graph serialized, for example, in the RDF/XML syntax, or an equivalent RDF graph serialization.

The package, though it does not contain a full SPARQL parser, makes an attempt to determine the query type when the query is set. This should work in most of the cases (but there is a possibility to set this manually, in case something goes wrong).

1.3.9 Automatic conversion of the results

To make processing somewhat easier, the package can do some conversions automatically from the return result. These are:

- for XML, the xml.dom.minidom is used to convert the result stream into a `Python representation of a DOM tree`.
- for JSON, the json package to generate a `Python dictionary`. Until version 1.3.1, the simplejson package was used.
- for CSV or TSV, a simple `string`.
- For RDF/XML and JSON-LD, the RDFLib package is used to convert the result into a `Graph` instance.
- For RDF Turtle/N3, a simple `string`.

There are two ways to generate this conversion:

- use `ret.convert()` in the return result from `sparql.query()` in the code above
- use `sparql.queryAndConvert()` to get the converted result right away if the intermediate stream is not used

For example, in the code below:

```
try :
    sparql.setReturnFormat(SPARQLWrapper.JSON)
    ret = sparql.query()
    dict = ret.convert()
except:
    deal_with_the_exception()
```

the value of `dict` is a Python dictionary of the query result, based on the SPARQL Query Results JSON Format.

1.3.10 Partial interpretation of the results

A further help is to offer an extra, partial interpretation of the results, again to cover most of the practical use cases. Based on the SPARQL Query Results JSON Format, the *SPARQLWrapper.SmartWrapper.Bindings* class can perform some simple steps in decoding the JSON return results. If *SPARQLWrapper.SmartWrapper.SPARQLWrapper2* is used instead of *SPARQLWrapper.Wrapper.SPARQLWrapper*, this result format is generated. Note that this relies on a JSON format only, ie, it has to be checked whether the SPARQL service can return JSON or not.

Here is a simple code that makes use of this feature:

```
from SPARQLWrapper import SPARQLWrapper2

queryString = "SELECT ?subj ?prop WHERE { ?subj ?prop ?o. }"

sparql = SPARQLWrapper2("http://example.org/sparql")

sparql.setQuery(queryString)
try :
    ret = sparql.query()
    print ret.variables  # this is an array consisting of "subj" and "prop"
    for binding in ret.bindings :
        # each binding is a dictionary. Let us just print the results
        print "%s: %s (of type %s)" % ("s",binding[u"subj"].value,binding[u"subj"].
↪type)
```

(continues on next page)

(continued from previous page)

```
        print "%s: %s (of type %s)" % ("p",binding[u"prop"].value,binding[u"prop"].
↪type)
except:
    deal_with_the_exception()
```

To make this type of code even easier to realize, the `[]` and `in` operators are also implemented on the result of *SPARQLWrapper.SmartWrapper.Bindings*. This can be used to check and find a particular binding (ie, particular row in the return value). This features becomes particularly useful when the `OPTIONAL` feature of SPARQL is used. For example:

```
from SPARQLWrapper import SPARQLWrapper2

queryString = "SELECT ?subj ?o ?opt WHERE { ?subj <http://a.b.c> ?o. OPTIONAL { ?subj
↪<http://d.e.f> ?opt }}"

sparql = SPARQLWrapper2("http://example.org/sparql")

sparql.setQuery(queryString)
try :
    ret = sparql.query()
    print ret.variables  # this is an array consisting of "subj", "o", "opt"
    if (u"subj",u"prop",u"opt") in ret :
       # there is at least one binding covering the optional "opt", too
       bindings = ret[u"subj",u"o",u"opt"]
       # bindings is an array of dictionaries with the full bindings
       for b in bindings :
           subj = b[u"subj"].value
           o    = b[u"o"].value
           opt  = b[u"opt"].value
           # do something nice with subj, o, and opt
    # another way of accessing to values for a single variable:
    # take all the bindings of the "subj"
    subjbind = ret.getValues(u"subj") # an array of Value instances
    ...
except:
    deal_with_the_exception()
```

1.3.11 GET or POST

By default, all SPARQL services are invoked using HTTP **GET** verb. However, **POST** might be useful if the size of the query extends a reasonable size; this can be set in the query instance.

Note that some combination may not work yet with all SPARQL processors (e.g., there are implementations where **POST + JSON return** does not work). Hopefully, this problem will eventually disappear.

1.4 Development

1.4.1 Requirements

The RDFLib package is used for RDF parsing.

This package is imported in a lazy fashion, ie, only when needed. Ie, if the user never intends to use the RDF format, the RDFLib package is not imported and the user does not have to install it.

1.4.2 Source code

The source distribution contains:

- `SPARQLWrapper`: the Python package. You should copy the directory somewhere into your PYTHONPATH. Alternatively, you can also run the distutils scripts: `python setup.py install`
- `test`: some unit and integrations tests. In order to run the tests some packages have to be installed before. So please install the packages listed in requirements.development.txt: `pip install -r requirements.development.txt`
- `scripts`: some scripts to run the package against some SPARQL endpoints.
- `docs`: the documentation.
- `custom_fixers`: 2to3 custom_fixer in order to fix an issue with urllib2._opener.

1.5 Community

Community support is available through the developer's discussion group rdflib-dev. The archives. from the old mailing list are still available.

1.6 Issues

Please, report any issue to github.

1.7 Documentation

The SPARQLWrapper documentation is available online.

Other interesting documents are the latest SPARQL 1.1 Specification (W3C Recommendation 21 March 2013) and the initial SPARQL Specification (W3C Recommendation 15 January 2008).

1.8 License

The SPARQLWrapper package is licensed under W3C license.

1.9 Acknowledgement

The package was greatly inspired by Lee Feigenbaum's similar package for Javascript.

Developers involved:

- Ivan Herman <http://www.ivan-herman.net>
- Sergio Fernández <http://www.wikier.org>
- Carlos Tejo Alonso <http://www.dayures.net>
- Alexey Zakhlestin <https://indeyets.ru/>

Organizations involved:

- World Wide Web Consortium
- Salzburg Research
- Foundation CTIC

CHAPTER 2

SPARQLWrapper package

2.1 SPARQLWrapper.Wrapper module

SPARQLWrapper.Wrapper.**XML = 'xml'**
to be used to set the return format to XML (SPARQL Query Results XML format or RDF/XML, depending on the query type). **This is the default**.

SPARQLWrapper.Wrapper.**JSON = 'json'**
to be used to set the return format to JSON.

SPARQLWrapper.Wrapper.**JSONLD = 'json-ld'**
to be used to set the return format to JSON-LD.

SPARQLWrapper.Wrapper.**TURTLE = 'turtle'**
to be used to set the return format to Turtle.

SPARQLWrapper.Wrapper.**N3 = 'n3'**
to be used to set the return format to N3 (for most of the SPARQL services this is equivalent to Turtle).

SPARQLWrapper.Wrapper.**RDF = 'rdf'**
to be used to set the return RDF Graph.

SPARQLWrapper.Wrapper.**RDFXML = 'rdf+xml'**
to be used to set the return format to RDF/XML explicitly.

SPARQLWrapper.Wrapper.**CSV = 'csv'**
to be used to set the return format to CSV

SPARQLWrapper.Wrapper.**TSV = 'tsv'**
to be used to set the return format to TSV

SPARQLWrapper.Wrapper.**GET = 'GET'**
to be used to set HTTP method GET. **This is the default**.

SPARQLWrapper.Wrapper.**POST = 'POST'**
to be used to set HTTP method POST.

SPARQLWrapper.Wrapper.**BASIC = 'BASIC'**
to be used to set BASIC HTTP Authentication method.

SPARQLWrapper.Wrapper.**DIGEST = 'DIGEST'**
to be used to set DIGEST HTTP Authentication method.

SPARQLWrapper.Wrapper.**SELECT = 'SELECT'**
to be used to set the query type to SELECT. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**CONSTRUCT = 'CONSTRUCT'**
to be used to set the query type to CONSTRUCT. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**ASK = 'ASK'**
to be used to set the query type to ASK. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**DESCRIBE = 'DESCRIBE'**
to be used to set the query type to DESCRIBE. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**INSERT = 'INSERT'**
to be used to set the query type to INSERT. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**DELETE = 'DELETE'**
to be used to set the query type to DELETE. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**CREATE = 'CREATE'**
to be used to set the query type to CREATE. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**CLEAR = 'CLEAR'**
to be used to set the query type to CLEAR. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**DROP = 'DROP'**
to be used to set the query type to DROP. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**LOAD = 'LOAD'**
to be used to set the query type to LOAD. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**COPY = 'COPY'**
to be used to set the query type to COPY. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**MOVE = 'MOVE'**
to be used to set the query type to MOVE. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**ADD = 'ADD'**
to be used to set the query type to ADD. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**URLENCODED = 'urlencoded'**
to be used to set **URL encode** as the encoding method for the request. This is, usually, determined automatically.

SPARQLWrapper.Wrapper.**POSTDIRECTLY = 'postdirectly'**
to be used to set **POST directly** as the encoding method for the request. This is, usually, determined automatically.

class SPARQLWrapper.Wrapper.**SPARQLWrapper**(*endpoint*, *updateEndpoint=None*, *returnFormat='xml'*, *defaultGraph=None*, *agent='sparqlwrapper 1.8.5 (rdflib.github.io/sparqlwrapper)'*)

Wrapper around an online access to a SPARQL Web entry point.

The same class instance can be reused for subsequent queries. The values of the base Graph URI, return formats, etc, are retained from one query to the next (in other words, only the query string changes). The instance can also be reset to its initial values using the *resetQuery()* method.

Variables

- **endpoint** (*string*) – SPARQL endpoint's URI.
- **updateEndpoint** (*string*) – SPARQL endpoint's URI for SPARQL Update operations (if it's a different one). The **default** value is `None`.
- **agent** (*string*) – The User-Agent for the HTTP request header. The **default** value is an autogenerated string using the SPARQLWrapper version code.
- **_defaultGraph** (*string*) – URI for the default graph. The value can be set either via an explicit call *addParameter("default-graph-uri", uri)* or as part of the query string. The **default** value is `None`.
- **user** (*string*) – The username of the credentials for querying the current endpoint. The value can be set an explicit call *setCredentials()*. The **default** value is `None`.
- **passwd** (*string*) – The password of the credentials for querying the current endpoint. The value can be set an explicit call *setCredentials()*. The **default** value is `None`.
- **http_auth** (*string*) – HTTP Authentication type. The **default** value is *BASIC*. Possible values are *BASIC* or *DIGEST*. It is used only in case the credentials are set.
- **onlyConneg** (*boolean*) – Option for allowing (or not) **only** HTTP Content Negotiation (so dismiss the use of HTTP parameters). The default value is `False`.
- **customHttpHeaders** (*dict*) – Custom HTTP Headers to be included in the request. It is a dictionary where keys are the header field and values are the header values. **Important**: These headers override previous values (including `Content-Type`, `User-Agent`, `Accept` and `Authorization` if they are present).
- **timeout** (*int*) – The timeout (in seconds) to use for querying the endpoint.
- **queryString** (*string*) – The SPARQL query text.
- **queryType** (*string*) – The type of SPARQL query (aka SPARQL query form), like *CONSTRUCT*, *SELECT*, *ASK*, *DESCRIBE*, *INSERT*, *DELETE*, *CREATE*, *CLEAR*, *DROP*, *LOAD*, *COPY*, *MOVE* or *ADD* (constants in this module).
- **returnFormat** (*string*) – The return format. No local check is done, so the parameter is simply sent to the endpoint. Eg, if the value is set to *JSON* and a construct query is issued, it is up to the endpoint to react or not, this wrapper does not check. The possible values are *JSON*, *XML*, *TURTLE*, *N3*, *RDF*, *RDFXML*, *CSV*, *TSV*, *JSONLD* (constants in this module). The **default** value is *XML*.
- **requestMethod** (*string*) – The request method for query or update operations. The possibles values are URL-encoded (*URLENCODED*) or POST directly (*POSTDIRECTLY*).
- **method** (*string*) – The invocation method (HTTP verb). The **default** value is *GET*, but it can be set to *POST*.
- **parameters** (*dict*) – The parameters of the request (key/value pairs in a dictionary).
- **_defaultReturnFormat** (*string*) – The default return format. It is used in case the same class instance is reused for subsequent queries.
- **prefix_pattern** (`re.RegexObject`, a compiled regular expression. See the `re` module of Python) – regular expression used to remove base/prefixes in the process of determining the query type.
- **pattern** (`re.RegexObject`, a compiled regular expression. See the `re` module of Python) – regular expression used to determine whether a query (without base/prefixes) is of type *CONSTRUCT*, *SELECT*, *ASK*, *DESCRIBE*, *INSERT*, *DELETE*, *CREATE*, *CLEAR*, *DROP*, *LOAD*, *COPY*, *MOVE* or *ADD*.

- **comments_pattern** (re.RegexObject, a compiled regular expression. See the re module of Python) – regular expression used to remove comments from a query.

__init__(*endpoint*, *updateEndpoint=None*, *returnFormat='xml'*, *defaultGraph=None*, *agent='sparqlwrapper 1.8.5 (rdflib.github.io/sparqlwrapper)'*)

Class encapsulating a full SPARQL call.

Parameters

- **endpoint** (*string*) – SPARQL endpoint's URI.
- **updateEndpoint** (*string*) – SPARQL endpoint's URI for update operations (if it's a different one). The **default** value is None.
- **returnFormat** – The return format. No local check is done, so the parameter is simply sent to the endpoint. Eg, if the value is set to *JSON* and a construct query is issued, it is up to the endpoint to react or not, this wrapper does not check. The possible values are *JSON*, *XML*, *TURTLE*, *N3*, *RDF*, *RDFXML*, *CSV*, *TSV*, *JSONLD* (constants in this module). The **default** value is *XML*.
- **defaultGraph** (*string*) – URI for the default graph. The value can be set either via an explicit call *addParameter("default-graph-uri", uri)* or as part of the query string. The **default** value is None.
- **agent** (*string*) – The User-Agent for the HTTP request header. The **default** value is an autogenerated string using the SPARQLWrapper version number.

resetQuery()

Reset the query, ie, return format, method, query, default or named graph settings, etc, are reset to their default values. This includes the default values for parameters, method, timeout or requestMethod.

setReturnFormat(*format*)

Set the return format. If the one set is not an allowed value, the setting is ignored.

Parameters **format** (*string*) – Possible values are *JSON*, *XML*, *TURTLE*, *N3*, *RDF*, *RDFXML*, *CSV*, *TSV*, *JSONLD* (constants in this module). All other cases are ignored.

Raises **ValueError** – If *JSONLD* is tried to set and the current instance does not support JSON-LD.

supportsReturnFormat(*format*)

Check if a return format is supported.

Parameters **format** (*string*) – Possible values are *JSON*, *XML*, *TURTLE*, *N3*, *RDF*, *RDFXML*, *CSV*, *TSV*, *JSONLD* (constants in this module). All other cases are ignored.

Returns Returns True if the return format is supported, otherwise False.

Return type bool

setTimeout(*timeout*)

Set the timeout (in seconds) to use for querying the endpoint.

Parameters **timeout** (*int*) – Timeout in seconds.

setOnlyConneg(*onlyConneg*)

Set this option for allowing (or not) only HTTP Content Negotiation (so dismiss the use of HTTP parameters).

New in version 1.8.1.

Parameters **onlyConneg** (*bool*) – True if **only** HTTP Content Negotiation is allowed; False if HTTP parameters are used.

setRequestMethod(*method*)
Set the internal method to use to perform the request for query or update operations, either URL-encoded (*URLENCODED*) or POST directly (*POSTDIRECTLY*). Further details at query operation in SPARQL and update operation in SPARQL Update.

Parameters **method** (*string*) – Possible values are *URLENCODED* (URL-encoded) or *POSTDIRECTLY* (POST directly). All other cases are ignored.

addDefaultGraph(*uri*)
Add a default graph URI.

Deprecated since version 1.6.0: Use *addParameter("default-graph-uri", uri)* instead of this method.

Parameters **uri** (*string*) – URI of the default graph.

addNamedGraph(*uri*)
Add a named graph URI.

Deprecated since version 1.6.0: Use *addParameter("named-graph-uri", uri)* instead of this method.

Parameters **uri** (*string*) – URI of the named graph.

addExtraURITag(*key*, *value*)
Some SPARQL endpoints require extra key value pairs. E.g., in virtuoso, one would add `should-sponge=soft` to the query forcing virtuoso to retrieve graphs that are not stored in its local database. Alias of *addParameter()* method.

Deprecated since version 1.6.0: Use *addParameter(key, value)* instead of this method

Parameters

- **key** (*string*) – key of the query part.
- **value** (*string*) – value of the query part.

addCustomParameter(*name*, *value*)
Method is kept for backwards compatibility. Historically, it "replaces" parameters instead of adding.

Deprecated since version 1.6.0: Use *addParameter(key, value)* instead of this method

Parameters

- **name** (*string*) – name.
- **value** (*string*) – value.

Returns Returns `True` if the adding has been accomplished, otherwise `False`.

Return type bool

addParameter(*name*, *value*)
Some SPARQL endpoints allow extra key value pairs. E.g., in virtuoso, one would add `should-sponge=soft` to the query forcing virtuoso to retrieve graphs that are not stored in its local database. If the parameter *query* is tried to be set, this intent is dismissed. Returns a boolean indicating if the set has been accomplished.

Parameters

- **name** (*string*) – name.
- **value** (*string*) – value.

Returns Returns `True` if the adding has been accomplished, otherwise `False`.

Return type bool

addCustomHttpHeader(*httpHeaderName*, *httpHeaderValue*)
Add a custom HTTP header (this method can override all HTTP headers).

Important: Take into account that each previous value for the header field names `Content-Type`, `User-Agent`, `Accept` and `Authorization` would be overriden if the header field name is present as value of the parameter `httpHeaderName`.

New in version 1.8.2.

Parameters

- **httpHeaderName** (*string*) – The header field name.
- **httpHeaderValue** (*string*) – The header field value.

clearCustomHttpHeader(*httpHeaderName*)
Clear the values of a custom HTTP Header previously set. Returns a boolean indicating if the clearing has been accomplished.

New in version 1.8.2.

Parameters **httpHeaderName** (*string*) – HTTP header name.

Returns Returns `True` if the clearing has been accomplished, otherwise `False`.

Return type bool

clearParameter(*name*)
Clear the values of a concrete parameter. Returns a boolean indicating if the clearing has been accomplished.

Parameters **name** (*string*) – name

Returns Returns `True` if the clearing has been accomplished, otherwise `False`.

Return type bool

setCredentials(*user*, *passwd*, *realm='SPARQL'*)
Set the credentials for querying the current endpoint.

Parameters

- **user** (*string*) – username.
- **passwd** (*string*) – password.
- **realm** (*string*) – realm. Only used for *DIGEST* authentication. The **default** value is `SPARQL`

Changed in version 1.8.3: Added `realm` parameter.

setHTTPAuth(*auth*)
Set the HTTP Authentication type. Possible values are *BASIC* or *DIGEST*.

Parameters **auth** (*string*) – auth type.

Raises

- **TypeError** – If the `auth` parameter is not an string.
- **ValueError** – If the `auth` parameter has not one of the valid values: *BASIC* or *DIGEST*.

setQuery(*query*)
Set the SPARQL query text.

Note: No check is done on the validity of the query (syntax or otherwise) by this module, except for testing the query type (SELECT, ASK, etc). Syntax and validity checking is done by the SPARQL service itself.

Parameters **query** (*string*) – query text.

Raises **TypeError** – If the *query* parameter is not an unicode-string or utf-8 encoded byte-string.

_parseQueryType(*query*)
Internal method for parsing the SPARQL query and return its type (ie, *SELECT*, *ASK*, etc).

Note: The method returns *SELECT* if nothing is specified. This is just to get all other methods running; in fact, this means that the query is erroneous, because the query must be, according to the SPARQL specification. The SPARQL endpoint should raise an exception (via `urllib`) for such syntax error.

Parameters **query** (*string*) – query text.

Returns the type of SPARQL query (aka SPARQL query form).

Return type string

setMethod(*method*)
Set the invocation method. By default, this is *GET*, but can be set to *POST*.

Parameters **method** (*string*) – should be either *GET* or *POST*. Other cases are ignored.

setUseKeepAlive()
Make `urllib2` use keep-alive.

Raises **ImportError** – when could not be imported `keepalive.HTTPHandler`.

isSparqlUpdateRequest()
Returns `True` if SPARQLWrapper is configured for executing SPARQL Update request.

Returns Returns `True` if SPARQLWrapper is configured for executing SPARQL Update request.

Return type bool

isSparqlQueryRequest()
Returns `True` if SPARQLWrapper is configured for executing SPARQL Query request.

Returns Returns `True` if SPARQLWrapper is configured for executing SPARQL Query request.

Return type bool

_cleanComments(*query*)
Internal method for returning the query after all occurrence of singleline comments are removed (issues #32 and #77).

Parameters **query** (*string*) – The query.

Returns the query after all occurrence of singleline comments are removed.

Return type string

_getRequestEncodedParameters(*query=None*)
Internal method for getting the request encoded parameters.

Parameters **query** (*tuple*) – a tuple of two items. The first item can be the string `query` (for *SELECT*, *DESCRIBE*, *ASK*, *CONSTRUCT* query) or the string `update` (for SPARQL Update queries, like *DELETE* or *INSERT*). The second item of the tuple is the query string itself.

Returns the request encoded parameters.

Return type string

_getAcceptHeader()
Internal method for getting the HTTP Accept Header.

See also:

Hypertext Transfer Protocol – HTTP/1.1 - Header Field Definitions

_createRequest()
Internal method to create request according a HTTP method. Returns a `urllib2.Request` object of the `urllib2` Python library

Raises **NotImplementedError** – If the HTTP authentification method is not one of the valid values: *BASIC* or *DIGEST*.

Returns request a `urllib2.Request` object of the `urllib2` Python library

_query()
Internal method to execute the query. Returns the output of the `urllib2.urlopen()` method of the `urllib2` Python library

Returns tuples with the raw request plus the expected format.

Raises

- ***QueryBadFormed*** – If the HTTP return code is `400`.
- ***Unauthorized*** – If the HTTP return code is `401`.
- ***EndPointNotFound*** – If the HTTP return code is `404`.
- ***URITooLong*** – If the HTTP return code is `414`.
- ***EndPointInternalError*** – If the HTTP return code is `500`.
- **urllib2.HTTPError** – If the HTTP return code is different to `400`, `401`, `404`, `414`, `500`.

query()
Execute the query. Exceptions can be raised if either the URI is wrong or the HTTP sends back an error (this is also the case when the query is syntactically incorrect, leading to an HTTP error sent back by the SPARQL endpoint). The usual urllib2 exceptions are raised, which therefore cover possible SPARQL errors, too.

Note that some combinations of return formats and query types may not make sense. For example, a SELECT query with Turtle response is meaningless (the output of a SELECT is not a Graph), or a CONSTRUCT query with JSON output may be a problem because, at the moment, there is no accepted JSON serialization of RDF (let alone one implemented by SPARQL endpoints). In such cases the returned media type of the result is unpredictable and may differ from one SPARQL endpoint implementation to the other. (Endpoints usually fall back to one of the "meaningful" formats, but it is up to the specific implementation to choose which one that is.)

Returns query result

Return type *QueryResult* instance

queryAndConvert()
Macro like method: issue a query and return the converted results.

Returns the converted query result. See the conversion methods for more details.

class SPARQLWrapper.Wrapper.**QueryResult**(*result*)
Wrapper around an a query result. Users should not create instances of this class, it is generated by a *SPARQLWrapper.query()* call. The results can be converted to various formats, or used directly.

If used directly: the class gives access to the direct HTTP request results `response` obtained from the call to `urllib.urlopen()`. It is a file-like object with two additional methods:

- `geturl()` to return the URL of the resource retrieved
- `info()` that returns the meta-information of the HTTP result as a dictionary-like object.

For convenience, these methods are also available on the *QueryResult* instance.

The `__iter__()` and *next()* methods are also implemented (by mapping them to `response`). This means that the common idiom `for l in obj : do_something_with_line(l)` would work, too.

Variables

- **response** – the direct HTTP response; a file-like object, as return by the `urllib2.urlopen()` library call.
- **requestedFormat** – The requested format. The possible values are: *JSON*, *XML*, *RDFXML*, *TURTLE*, *N3*, *RDF*, *CSV*, *TSV*, *JSONLD*.

__init__(*result*)

Parameters **result** – HTTP response stemming from a *SPARQLWrapper.query()* call, or a tuple with the expected format: (response, format).

geturl()
Return the URL of the original call.

Returns URL of the original call.

Return type string

info()
Return the meta-information of the HTTP result.

Returns meta-information of the HTTP result.

Return type dict

next()
Method for the standard iterator.

_convertJSON()
Convert a JSON result into a Python dict. This method can be overwritten in a subclass for a different conversion method.

Returns converted result.

Return type dict

_convertXML()
Convert an XML result into a Python dom tree. This method can be overwritten in a subclass for a different conversion method.

Returns converted result.

Return type xml.dom.minidom.Document

_convertRDF()
Convert a RDF/XML result into an RDFLib Graph. This method can be overwritten in a subclass for a different conversion method.

Returns converted result.

Return type rdflib.graph.Graph

_convertN3()
Convert a RDF Turtle/N3 result into a string. This method can be overwritten in a subclass for a different conversion method.

Returns converted result.

Return type string

_convertCSV()
Convert a CSV result into a string. This method can be overwritten in a subclass for a different conversion method.

Returns converted result.

Return type string

_convertTSV()
Convert a TSV result into a string. This method can be overwritten in a subclass for a different conversion method.

Returns converted result.

Return type string

_convertJSONLD()
Convert a RDF JSON-LD result into an RDFLib Graph. This method can be overwritten in a subclass for a different conversion method.

Returns converted result

Return type rdflib.graph.Graph

convert()
Encode the return value depending on the return format:

- in the case of *XML*, a DOM top element is returned
- in the case of *JSON*, a json conversion will return a dictionary
- in the case of *RDF/XML*, the value is converted via RDFLib into a RDFLib Graph instance
- in the case of *JSON-LD*, the value is converted via RDFLib into a RDFLib Graph instance
- in the case of RDF *Turtle*/*N3*, a string is returned
- in the case of *CSV*/*TSV*, a string is returned
- In all other cases the input simply returned.

Returns the converted query result. See the conversion methods for more details.

_get_responseFormat()
Get the response (return) format. The possible values are: *JSON*, *XML*, *RDFXML*, *TURTLE*, *N3*, *CSV*, *TSV*, *JSONLD*. In case there is no Content-Type, None is return. In all other cases, the raw Content-Type is return.

New in version 1.8.3.

Returns the response format. The possible values are: *JSON*, *XML*, *RDFXML*, *TURTLE*, *N3*, *CSV*, *TSV*, *JSONLD*.

Return type string

print_results(*minWidth=None*)

This method prints a representation of a *QueryResult* object that MUST has as response format *JSON*.

Parameters **minWidth** (*string*) – The minimum width, counting as characters. The default value is None.

2.2 SPARQLWrapper.SmartWrapper module

class SPARQLWrapper.SmartWrapper.**Bindings**(*retval*)

Bases: object

Class encapsulating one query result, based on the JSON return format. It decodes the return values to make it a bit more usable for a standard usage. The class consumes the return value and instantiates a number of attributes that can be consulted directly. See the list of variables.

The Serializing SPARQL Query Results in JSON explains the details of the JSON return structures. Very succinctly: the return data has "bindings", which means a list of dictionaries. Each dictionary is a possible binding of the SELECT variables to *Value* instances. This structure is made a bit more usable by this class.

Variables

- **fullResult** (*dict*) – The original dictionary of the results, stored for an easier reference.
- **head** (*dict*) – Header part of the return, see the JSON return format document for details.
- **variables** (*list*) – List of unbounds (variables) of the original query. It is a list of strings. None in the case of an ASK query.
- **bindings** (*list*) – The final bindings: list of dictionaries, mapping variables to *Value* instances. If unbound, then no value is set in the dictionary; that can be easily checked with var in res.bindings[..], for example.
- **askResult** (*bool*) – by default, set to **False**; in case of an ASK query, the result of the query.

__init__(*retval*)

Parameters **retval** (*QueryResult*) – the query result.

convert()

This is just a convenience method, returns self.

Although SPARQLWrapper2.Bindings is not a subclass of *SPARQLWrapper.QueryResult*, it is returned as a result by *SPARQLWrapper2.query()*, just like *QueryResult* is returned by *SPARQLWrapper.query()*. Consequently, having an empty *convert()* method to imitate *QueryResult's convert() method* may avoid unnecessary problems.

getValues(*key*)

A shorthand for the retrieval of all bindings for a single key. It is equivalent to [b[key] for b in self[key]]

Parameters **key** (*string*) – possible variable name.

Returns list of *Value* instances.

Return type list

class SPARQLWrapper.SmartWrapper.**SPARQLWrapper2**(*baseURI*, *defaultGraph=None*)
Bases: *SPARQLWrapper.Wrapper.SPARQLWrapper*

Subclass of *SPARQLWrapper* that works with a JSON SELECT return result only. The query result is automatically set to a *Bindings* instance. Makes the average query processing a bit simpler...

__init__(*baseURI*, *defaultGraph=None*)
Class encapsulating a full SPARQL call. In contrast to the *SPARQLWrapper* superclass, the return format cannot be set (it is defaulted to JSON).

Parameters

- **baseURI** (*string*) – string of the SPARQL endpoint's URI.
- **defaultGraph** (*string*) – URI for the default graph. Default is None, can be set via an explicit call, too.

query()
Execute the query and do an automatic conversion.

Exceptions can be raised if either the URI is wrong or the HTTP sends back an error. The usual urllib2 exceptions are raised, which cover possible SPARQL errors, too.

If the query type is *not* SELECT, the method falls back to the *corresponding method in the superclass*.

Returns query result

Return type *Bindings* instance

queryAndConvert()
This is here to override the inherited method; it is equivalent to *query*.

If the query type is *not* SELECT, the method falls back to the *corresponding method in the superclass*.

Returns the converted query result.

setReturnFormat(*format*)
Set the return format (*overriding the inherited method*).

> **Warning:** This method does nothing; this class instance should work with JSON only. The method is defined just to avoid possible errors by erroneously setting the return format. When using this class, the user can safely ignore this call.

Parameters **format** (*string*) – return format

class SPARQLWrapper.SmartWrapper.**Value**(*variable*, *binding*)
Bases: object

Class encapsulating a single binding for a variable.

Variables

- **variable** (*string*) – The original variable, stored for an easier reference.
- **value** (*string*) – Value of the binding.
- **type** (*string*) – Type of the binding. One of *Value.URI*, *Value.Literal*, *Value.TypedLiteral*, or *Value.BNODE*.

- **lang** (*string*) – Language tag of the binding, or `None` if not set.
- **datatype** (*string*) – Datatype of the binding, or `None` if not set. It is an URI.

BNODE = 'bnode'
the string denoting a blank node variable.

Literal = 'literal'
the string denoting a Literal variable.

TypedLiteral = 'typed-literal'
the string denoting a typed literal variable.

URI = 'uri'
the string denoting a URI variable.

__init__(*variable*, *binding*)

Parameters

- **variable** (*string*) – the variable for that binding. Stored for an easier reference.
- **binding** (*dict*) – the binding dictionary part of the return result for a specific binding.

2.3 SPARQLWrapper.SPARQLExceptions module

SPARQL Wrapper exceptions

exception SPARQLWrapper.SPARQLExceptions.**SPARQLWrapperException**(*response=None*)
Bases: exceptions.Exception

Base class for SPARQL Wrapper exceptions

__init__(*response=None*)

Parameters **response** (*string*) – The server response

exception SPARQLWrapper.SPARQLExceptions.**EndPointInternalError**(*response=None*)
Bases: *SPARQLWrapper.SPARQLExceptions.SPARQLWrapperException*

Exception type for Internal Server Error responses. Usually HTTP response status code `500`.

exception SPARQLWrapper.SPARQLExceptions.**QueryBadFormed**(*response=None*)
Bases: *SPARQLWrapper.SPARQLExceptions.SPARQLWrapperException*

Query Bad Formed exception. Usually HTTP response status code `400`.

exception SPARQLWrapper.SPARQLExceptions.**EndPointNotFound**(*response=None*)
Bases: *SPARQLWrapper.SPARQLExceptions.SPARQLWrapperException*

End Point Not Found exception. Usually HTTP response status code `404`.

exception SPARQLWrapper.SPARQLExceptions.**Unauthorized**(*response=None*)
Bases: *SPARQLWrapper.SPARQLExceptions.SPARQLWrapperException*

Access is denied due to invalid credentials (unauthorized). Usually HTTP response status code `401`.

New in version 1.8.2.

exception SPARQLWrapper.SPARQLExceptions.**URITooLong**(*response=None*)
Bases: *SPARQLWrapper.SPARQLExceptions.SPARQLWrapperException*

The URI requested by the client is longer than the server is willing to interpret. Usually HTTP response status code `414`.

New in version 1.8.3.

2.4 SPARQLWrapper.KeyCaseInsensitiveDict module

A simple implementation of a key case-insensitive dictionary.

class SPARQLWrapper.KeyCaseInsensitiveDict.**KeyCaseInsensitiveDict**(*d={}*)

Bases: dict

A simple implementation of a key case-insensitive dictionary

__init__(*d={}*)

Parameters **d** (*dict*) – The source dictionary.

CHAPTER 3

SPARQLWrapper's changelog

3.1 2019-12-22 1.8.5

- Improve/tests for development (#131)
- Changed. Be more strict on Accept Turtle header (#137)
- Migrated documentation from epydoc to sphinx and readthedocs

3.2 2019-04-18 1.8.4

- Added example
- hotfix: Added custom_fixers folder in MANIFEST, in order to be used in python3 (#129)

3.3 2019-04-17 1.8.3

- Include ChangeLog.txt in the distribution
- Removed import of SPARQLWrapper in setup.py (fixed #113 and closed #115)
- Added support for querying RDF/XML in a CONSTRUCT query type
- Updated the procedure for determining the query type (#120)
- Do not send format parameter for the results ([format, output, results]) when the query is a SPARQL Update query
- Added test for new agrovoc SPARQL endpoint (using Fuseki2)
- Added test for 4store SPARQL endpoint (used by agroportal)
- Added/Updated tests

- Added examples
- Updated doc
- Fixed code generated for python3 using 2to3, adding a custom fixer (#109)

3.4 2018-05-26 1.8.2

- Fixed bug (#100)
- Updated doc
- Added Unauthorized exception in SPARQLWrapperExceptions
- Added support for custom HTTP headers (#52)
- Changed timeout setting (#106)

3.5 2018-02-25 1.8.1

- Update classifiers (Python 3.6)
- Added some documentation about the parameter to indicate the output format
- Fixed typo in width calculation
- Added support for CSV, TSV (PR #98)
- Added support for Only HTTP Content Negotiation (#82)

3.6 2016-12-07 1.8.0

- Updated return formats for not content negotiation situations
- Included license in the MANIFEST (issue #76)
- Added explicit support for RDF/XML as allowed format (issue #75)
- Added proper shebang (issue #78)
- Moved keepalive as optional dependency (issue #79)
- Fixed hash check on prefixes (issue #77)
- Fixed epydoc warnings (issue #41)

3.7 2015-12-18 1.7.6

- Removed wrong response encoding (issue #70)
- Authorization header bug when using Python 3 (issue #71)

3.8 2015-11-19 1.7.5

- Removed pip dependency on setup (issue #69)

3.9 2015-11-05 1.7.4

- Fixed packaging (issue #66)

3.10 2015-11-05 1.7.3

- Finally fixed the keepalive issue in all Pyhon versions (issue #65)
- Removed old JSON layer in favor of the native json module

3.11 2015-11-03 1.7.2

- Moved to the new keepalive package (issues #53 and #61)

3.12 2015-10-29 1.7.1

- Fixed build in Python 3.x (issue #57)

3.13 2015-10-29 1.7.0

- Added support to HTTP Digest Auth Support (issue #45)
- Improved print_results showing language tag (xml:lang) and datatype
- Updated to RDFLib 4.x

3.14 2014-08-26 1.6.4

- Fixed unicode problems on setup (issue #42)

3.15 2014-08-26 1.6.3

- Fixed unicode problems with urllib in Python 3 (issue #35)
- Restored SPARQLWrapper2 class (issue #36)
- Enhanced warning for missing rdflib-jsonld (issue #38)
- Fixed build system (issue #39)

3.16 2014-07-24 1.6.2

- Fixed query type detection with comments (issue #32)

3.17 2014-07-21 1.6.1

- Added missing query types (issue #17)
- Added a new method to the API to select the request method to be fully SPARQL 1.1 Protocol compliant (issue #28)
- Improved the test suite coverage, including support to run the tests under Python 3.x (issues #20, #24 and #31)

3.18 2014-05-09 1.6.0

- Returning raw response in case of unknown content type returned
- Fixed some issues with the last version of the SPARQL 1.1 Update Protocol
- setQuery() doesn't imply resetQuery() anymore
- Deprecated addCustomParameter(), addParameter() and clearParameter() come to provide all required functionality
- SPARQLWrapper, QueryResult, Value, Bindings (and classes inherited from them) are new-style classes now
- POST queries are accompanied by full set of parameters now
- Added rudimentary support for JSON-LD
- Added proper unit tests without dependencies of external endpoints
- Fixed Python 3 compatibility issues in SmartWrapper module

3.19 2012-08-28 1.5.2

- Implemented update operation according the latest SPARQL 1.1 Protocol drafts (i.e., switching to 'update' parameter)

3.20 2012-07-10 1.5.1

- Added the possibility to use two different endpoints for reading and writing operations
- New print_results() function for users testing

3.21 2012-02-01 1.5.0

- Update handling 500's coming from SPARQL endpoint (feature request #3198363)
- Added Python 3.x support (feature request 3022722)
- Warning when returned format would be different than the requested one

3.22 2011-01-28 1.4.2

- Updated for working with RDFLib3 too (feature request #3117442)
- fixed bug with prefixes' regex (#2320024)

3.23 2010-01-11 1.4.1

- Supporting keep-alive in SPARQLWrapper if urlgrabber is available (ticket #2929881)
- fixed bugs (#2949834)

3.24 2009-12-14 1.4.0

- Added some support for SPARUL
- Improved HTTP related code
- Many other minor bugs fixed

3.25 2009-09-23 1.3.2

- Remove pyxml dependency. Instead, use xml.dom.minidom
- Updated setup installation (added rdflib dependency)
- Updated example.py (added XML, N3 and RDF examples)

3.26 2009-09-11 1.3.1

- Remove simplejson dependency for python => 2.6 version
- Added feature to choose the json module to use

3.27 2009-05-06 1.3.0

- Added a new method to add custom parameters (deprecated old way to do it)

3.28 2009-04-27 1.2.1

- Updated setup installation
- Patched to work with JSON in Python>=2.6

3.29 2008-07-10 1.2.0

- Allowed non-standard extensions (such as SPARUL).
- Exceptions fixed.
- Added another example.

3.30 2008-03-24 1.1.0

- Renamed package name to SPARQLWrapper.
- Added a basic catalog of exceptions.

3.31 2008-03-07 1.0.1

- Fixed some cosmetic things.

3.32 2008-02-14 1.0.0

- First stable release.
- Main functionality stabilized.
- Project moved to SourceForge.

3.33 2007-07-06 0.2.0

- First public release of the library.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

B

T

U

V

X