SPARQLWrapper Documentation

CHANGE_AUTHORS

Mar 13, 2022

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 ONE

SPARQL ENDPOINT INTERFACE TO PYTHON

1.1 About

SPARQLWrapper is a simple Python wrapper around a SPARQL service to remotely execute your queries. It helps by creating the query invocation and, optionally, converting 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

You can use SPARQLWrapper either as a Python command line script or as a Python package.

1.3.1 Command Line Script

To use as a command line script, you will need to install SPARQLWrapper and then a command line script called `rqw` (spaRQl Wrapper) will be available within the Python environment into which it is installed. run `$ rql -h` to see all the script's options.

1.3.2 Python package

Here are a series of examples of different queries executed via SPARQLWrapper as a python package.

SELECT examples

Simple use of this module is as follows where a live SPARQL endpoint is given and the JSON return format is used:

```
from SPARQLWrapper import SPARQLWrapper, JSON

sparql = SPARQLWrapper(
    "http://vocabs.ardc.edu.au/repository/api/sparql/"
    "csiro_international-chronostratigraphic-chart_geologic-time-scale-2020"
)
sparql.setReturnFormat(JSON)

# gets the first 3 geological ages
# from a Geological Timescale database,
# via a SPARQL endpoint
sparql.setQuery("""
    PREFIX gts: <http://resource.geosciml.org/ontology/timescale/gts#>

    SELECT *
    WHERE {
        ?a a gts:Age .
    }
    ORDER BY ?a
    LIMIT 3
    """
)

try:
    ret = sparql.queryAndConvert()

    for r in ret["results"]["bindings"]:
        print(r)
except Exception as e:
    print(e)
```

This should print out something like this:

```
{'a': {'type': 'uri', 'value': 'http://resource.geosciml.org/classifier/ics/ischart/
↪Aalenian'}}
{'a': {'type': 'uri', 'value': 'http://resource.geosciml.org/classifier/ics/ischart/
↪Aeronian'}}
{'a': {'type': 'uri', 'value': 'http://resource.geosciml.org/classifier/ics/ischart/
↪Albian'}}
```

The above result is the response from the given endpoint, retrieved in JSON, and converted to a Python object, `ret`, which is then iterated over and printed.

ASK example

This query gets a boolean response from DBPedia's SPARQL endpoint:

```python
from SPARQLWrapper import SPARQLWrapper, XML

sparql = SPARQLWrapper("http://dbpedia.org/sparql")
sparql.setQuery("""
    ASK WHERE {
        <http://dbpedia.org/resource/Asturias> rdfs:label "Asturias"@es
    }
""")
sparql.setReturnFormat(XML)
results = sparql.query().convert()
print(results.toxml())
```

You should see something like:

```
<?xml version="1.0" ?>
<sparql
    xmlns="http://www.w3.org/2005/sparql-results#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd">
<head/>
    <boolean>true</boolean>
</sparql>
```

CONSTRUCT example

CONSTRUCT queries return RDF, so `queryAndConvert()` here produces an RDFlib `Graph` object which is then serialized to the Turtle format for printing:

```python
from SPARQLWrapper import SPARQLWrapper

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

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

    CONSTRUCT {
```

(continues on next page)

(continued from previous page)

```
      ?lang a sdo:Language ;
      sdo:alternateName ?iso6391Code .
    }
    WHERE {
      ?lang a dbo:Language ;
      dbo:iso6391Code ?iso6391Code .
      FILTER (STRLEN(?iso6391Code)=2) # to filter out non-valid values
    }
    LIMIT 3
""")

results = sparql.queryAndConvert()
print(results.serialize())
```

Results from this query should look something like this:

```
@prefix schema: <https://schema.org/> .

<http://dbpedia.org/resource/Arabic> a schema:Language ;
    schema:alternateName "ar" .

<http://dbpedia.org/resource/Aragonese_language> a schema:Language ;
    schema:alternateName "an" .

<http://dbpedia.org/resource/Uruguayan_Spanish> a schema:Language ;
    schema:alternateName "es" .
```

DESCRIBE example

Like CONSTRUCT queries, DESCRIBE queries also produce RDF results, so this example produces an RDFlib `Graph` object which is then serialized into the JSON-LD format and printed:

```
from SPARQLWrapper import SPARQLWrapper

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

results = sparql.queryAndConvert()
print(results.serialize(format="json-ld"))
```

The result for this example is large but starts something like this:

```
[
    {
        "@id": "http://dbpedia.org/resource/Mazonovo",
        "http://dbpedia.org/ontology/subdivision": [
            {
                "@id": "http://dbpedia.org/resource/Asturias"
            }
    ],
...
```

SPARQL UPDATE example

UPDATE queries write changes to a SPARQL endpoint, so we can't easily show a working example here. However, if `https://example.org/sparql` really was a working SPARQL endpoint that allowed updates, the following code might work:

```
from SPARQLWrapper import SPARQLWrapper, POST, DIGEST

sparql = SPARQLWrapper("https://example.org/sparql")
sparql.setHTTPAuth(DIGEST)
sparql.setCredentials("some-login", "some-password")
sparql.setMethod(POST)

sparql.setQuery("""
    PREFIX dbp:  <http://dbpedia.org/resource/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

    WITH <http://example.graph>
    DELETE {
       dbo:Asturias rdfs:label "Asturies"@ast
    }
    """
)

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

If the above code really worked, it would delete the triple `dbo:Asturias rdfs:label "Asturies"@ast` from the graph `http://example.graph`.

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 even simpler.

```
from SPARQLWrapper import SPARQLWrapper2

sparql = SPARQLWrapper2("http://dbpedia.org/sparql")
sparql.setQuery("""
    PREFIX dbp:  <http://dbpedia.org/resource/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

    SELECT ?label
    WHERE {
        dbp:Asturias rdfs:label ?label
    }
    LIMIT 3
    """
                )

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

The above should print out something like:

```
en, Asturias
ar,
ca, Astúries
```

1.3.3 Return formats

The expected return formats differs per 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.

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`.
- 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()
    d = ret.convert()
except Exception as e:
    print(e)
```

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

Partial interpretation of the results

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:

```python
from SPARQLWrapper import SPARQLWrapper2

sparql = SPARQLWrapper2("http://example.org/sparql")
sparql.setQuery("""
    SELECT ?subj ?prop
    WHERE {
        ?subj ?prop ?obj
    }
    """
)

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(f"{binding['subj'].value}, {binding['subj'].type}")
        print(f"{binding['prop'].value}, {binding['prop'].type}")
except Exception as e:
    print(e)
```

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:

```python
from SPARQLWrapper import SPARQLWrapper2

sparql = SPARQLWrapper2("http://example.org/sparql")
sparql.setQuery("""
    SELECT ?subj ?obj ?opt
    WHERE {
        ?subj <http://a.b.c> ?obj .
        OPTIONAL {
            ?subj <http://d.e.f> ?opt
        }
    }
    """
)

try:
    ret = sparql.query()
    print(ret.variables)  # this is an array consisting of "subj", "obj", "opt"
    if ("subj", "prop", "opt") in ret:
```

(continues on next page)

(continued from previous page)

```
        # there is at least one binding covering the optional "opt", too
        bindings = ret["subj", "obj", "opt"]
        # bindings is an array of dictionaries with the full bindings
        for b in bindings:
            subj = b["subj"].value
            o = b["obj"].value
            opt = b["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("subj")  # an array of Value instances
    ...
except Exception as e:
    print(e)
```

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 combinations 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 SPARQL Endpoint Implementations

1.4.1 Introduction

From SPARQL 1.1 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 fact is that the **parameter key** for the choice of the **output format** is not defined. Virtuoso uses *format*, Fuseki uses *output*, rasqual seems to use *results*, etc… Also, in some cases HTTP Content Negotiation can/must be used.

1.4.2 ClioPatria

Website The SWI-Prolog Semantic Web Server

Documentation Search ‘sparql’ in http://cliopatria.swi-prolog.org/help/http.

Uses Parameters **and** Content Negotiation.

Parameter key `format`.

Parameter value MUST be one of these values: `rdf+xml`, `json`, `csv`, `application/sparql-results+xml` or `application/sparql-results+json`.

1.4.3 OpenLink Virtuoso

Website OpenLink Virtuoso

Parameter key `format` or `output`.

JSON-LD (application/ld+json) supported (in CONSTRUCT and DESCRIBE).

- Parameter value, like directly: "text/html" (HTML), "text/x-html+tr" (HTML (Faceted Browsing Links)), "application/vnd.ms-excel", "application/sparql-results+xml" (XML), "application/sparql-results+json" (JSON), "application/javascript" (Javascript), "text/turtle" (Turtle), "application/rdf+xml" (RDF/XML), "text/plain" (N-Triples), "text/csv" (CSV), "text/tab-separated-values" (TSV)
- Parameter value, like indirectly: "HTML" (alias text/html), "JSON" (alias application/sparql-results+json), "XML" (alias application/sparql-results+xml), "TURTLE" (alias text/rdf+n3), JavaScript (alias application/javascript) See http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSSparqlProtocol#AdditionalHTTPResponseFormats–SELECT
- For a `SELECT` query type, the default return mimetype (if `Accept: */*` is sent) is `application/sparql-results+xml`
- For a `ASK` query type, the default return mimetype (if `Accept: */*` is sent) is `text/html`
- For a `CONSTRUCT` query type, the default return mimetype (if `Accept: */*` is sent) is `text/turtle`
- For a `DESCRIBE` query type, the default return mimetype (if `Accept: */*` is sent) is `text/turtle`

1.4.4 Fuseki

Website Fuseki

Uses Parameters **and** Content Negotiation.

Parameter key `format` or `output` (Fuseki 1, Fuseki 2).

JSON-LD (application/ld+json) supported (in CONSTRUCT and DESCRIBE).

- Fuseki 1 - Short names for "output=" : "json", "xml", "sparql", "text", "csv", "tsv", "thrift"
- Fuseki 2 - Short names for "output=" : "json", "xml", "sparql", "text", "csv", "tsv", "thrift"
- If a non-expected short name is used, the server returns an "Error 400: Can't determine output serialization"
- Valid alias for SELECT and ASK: "json", "xml", csv", "tsv"
- Valid alias for DESCRIBE and CONSTRUCT: "json" (alias for json-ld ONLY in Fuseki 2), "xml"
- Valid mimetype for DESCRIBE and CONSTRUCT: "application/ld+json"
- Default return mimetypes: For a SELECT and ASK query types, the default return mimetype (if Accept: */* is sent) is application/sparql-results+json
- Default return mimetypes: For a DESCRIBE and CONTRUCT query types, the default return mimetype (if Accept: */* is sent) is text/turtle
- In case of a bad formed query, Fuseki 1 returns 200 instead of 400.

1.4.5 Eclipse RDF4J

Website Eclipse RDF4J (formerly known as OpenRDF Sesame)

Documentation https://rdf4j.eclipse.org/documentation/rest-api/#the-query-operation, https://rdf4j.eclipse.org/documentation/rest-api/#content-types

Uses Only content negotiation (no URL parameters).

Parameter If an unexpected parameter is used, the server ignores it.

JSON-LD (application/ld+json) supported (in CONSTRUCT and DESCRIBE).

- SELECT
  - `application/sparql-results+xml` (DEFAULT if `Accept: */*` is sent))
  - `application/sparql-results+json` (also `application/json`)
  - `text/csv`
  - `text/tab-separated-values`
  - Other values: `application/x-binary-rdf-results-table`
- ASK
  - `application/sparql-results+xml` (DEFAULT if `Accept: */*` is sent))
  - `application/sparql-results+json`
  - Other values: `text/boolean`
  - **Not supported**: `text/csv`
  - **Not supported**: `text/tab-separated-values`
- CONSTRUCT
  - `application/rdf+xml`
  - `application/n-triples` (DEFAULT if `Accept: */*` is sent)
  - `text/turtle`
  - `text/n3`
  - `application/ld+json`
  - Other acceptable values: `application/n-quads`, `application/rdf+json`, `application/trig`, `application/trix`, `application/x-binary-rdf`
  - `text/plain` (returns `application/n-triples`)
  - `text/rdf+n3` (returns `text/n3`)
  - `text/x-nquads` (returns `application/n-quads`)
- DESCRIBE
  - `application/rdf+xml`
  - `application/n-triples` (DEFAULT if `Accept: */*` is sent)
  - `text/turtle`
  - `text/n3`
  - `application/ld+json`

  - Other acceptable values: `application/n-quads`, `application/rdf+json`, `application/trig`, `application/trix`, `application/x-binary-rdf`
  - `text/plain` (returns `application/n-triples`)
  - `text/rdf+n3` (returns `text/n3`)
  - `text/x-nquads` (returns `application/n-quads`)

1.4.6 RASQAL

Website RASQAL

Documentation http://librdf.org/rasqal/roqet.html

Parameter key `results`.

JSON-LD (application/ld+json) NOT supported.

Uses roqet as RDF query utility (see http://librdf.org/rasqal/roqet.html) For variable bindings, the values of FORMAT vary upon what Rasqal supports but include simple for a simple text format (default), xml for the SPARQL Query Results XML format, csv for SPARQL CSV, tsv for SPARQL TSV, rdfxml and turtle for RDF syntax formats, and json for a JSON version of the results.

For RDF graph results, the values of FORMAT are ntriples (N-Triples, default), rdfxml-abbrev (RDF/XML Abbreviated), rdfxml (RDF/XML), turtle (Turtle), json (RDF/JSON resource centric), json-triples (RDF/JSON triples) or rss-1.0 (RSS 1.0, also an RDF/XML syntax).

1.4.7 Marklogic

Website Marklogic

Uses Only content negotiation (no URL parameters).

JSON-LD (application/ld+json) NOT supported.

You can use following methods to query triples:

- SPARQL mode in Query Console. For details, see Querying Triples with SPARQL
- XQuery using the semantics functions, and Search API, or a combination of XQuery and SPARQL. For details, see Querying Triples with XQuery or JavaScript.
- HTTP via a SPARQL endpoint. For details, see Using Semantics with the REST Client API.

Formats are specified as part of the HTTP Accept headers of the REST request. When you query the SPARQL endpoint with REST Client APIs, you can specify the result output format (See https://docs.marklogic.com/guide/semantics/REST#id_54258. The response type format depends on the type of query and the MIME type in the HTTP Accept header.

This table describes the MIME types and Accept Header/Output formats (MIME type) for different types of SPARQL queries. (See https://docs.marklogic.com/guide/semantics/REST#id_54258 and https://docs.marklogic.com/guide/semantics/loading#id_70682)

- SELECT
  - application/sparql-results+xml
  - application/sparql-results+json
  - text/html
  - text/csv

- ASK queries return a boolean (true or false).
- CONSTRUCT or DESCRIBE
  - application/n-triples
  - application/rdf+json
  - application/rdf+xml
  - text/turtle
  - text/n3
  - application/n-quads
  - application/trig

1.4.8 AllegroGraph

Website AllegroGraph

Documentation https://franz.com/agraph/support/documentation/current/http-protocol.html

Uses Only content negotiation (no URL parameters).

Parameter The server always looks at the Accept header of a request, and tries to generate a response in the format that the client asks for. If this fails, a 406 response is returned. When no Accept, or an Accept of / is specified, the server prefers text/plain, in order to make it easy to explore the interface from a web browser.

JSON-LD (application/ld+json) NOT supported.

- SELECT
  - application/sparql-results+xml (DEFAULT if Accept: / is sent)
  - application/sparql-results+json (and application/json)
  - text/csv
  - text/tab-separated-values
  - OTHERS: application/sparql-results+ttl, text/integer, application/x-lisp-structured-expression, text/table, application/processed-csv, text/simple-csv, application/x-direct-upis
- ASK
  - application/sparql-results+xml (DEFAULT if Accept: / is sent)
  - application/sparql-results+json (and application/json)
  - Not supported: text/csv
  - Not supported: text/tab-separated-values
- CONSTRUCT
  - application/rdf+xml (DEFAULT if Accept: / is sent)
  - text/rdf+n3
  - OTHERS: text/integer, application/json, text/plain, text/x-nquads, application/trix, text/table, application/x-direct-upis
- DESCRIBE
  - application/rdf+xml (DEFAULT if Accept: / is sent)

  - text/rdf+n3

1.4.9 4store

Website 4store

Documentation https://4store.danielknoell.de/trac/wiki/SparqlServer/

Uses Parameters **and** Content Negotiation.

Parameter key `output`.

Parameter value alias. If an unexpected alias is used, the server is not working properly.

JSON-LD (application/ld+json) NOT supported.

- SELECT
  - application/sparql-results+xml (alias xml) (DEFAULT if Accept: / is sent))
  - application/sparql-results+json or application/json (alias json)
  - text/csv (alias csv)
  - text/tab-separated-values (alias tsv). Returns "text/plain" in GET.
  - Other values: text/plain, application/n-triples
- ASK
  - application/sparql-results+xml (alias xml) (DEFAULT if Accept: / is sent))
  - application/sparql-results+json or application/json (alias json)
  - text/csv (alias csv)
  - text/tab-separated-values (alias tsv). Returns "text/plain" in GET.
  - Other values: text/plain, application/n-triples
- CONSTRUCT
  - application/rdf+xml (alias xml) (DEFAULT if Accept: / is sent)
  - text/turtle (alias "text")
- DESCRIBE
  - application/rdf+xml (alias xml) (DEFAULT if Accept: / is sent)
  - text/turtle (alias "text")

Valid alias for SELECT and ASK "json", "xml", csv", "tsv" (also "text" and "ascii")

Valid alias for DESCRIBE and CONSTRUCT "xml", "text" (for turtle)

1.4.10 Blazegraph

Website Blazegraph (Formerly known as Bigdata) & NanoSparqlServer

Documentation https://wiki.blazegraph.com/wiki/index.php/REST_API#SPARQL_End_Point

Uses Parameters **and** Content Negotiation.

Parameter key `format` (available since version 1.4.0). Setting this parameter will override any Accept Header that is present

Parameter value alias. If an unexpected alias is used, the server is not working properly.

JSON-LD (application/ld+json) NOT supported.

- SELECT
  - application/sparql-results+xml (alias xml) (DEFAULT if Accept: / is sent))
  - application/sparql-results+json or application/json (alias json)
  - text/csv
  - text/tab-separated-values
  - Other values: application/x-binary-rdf-results-table
- ASK
  - application/sparql-results+xml (alias xml) (DEFAULT if Accept: / is sent))
  - application/sparql-results+json or application/json (alias json)
- CONSTRUCT
  - application/rdf+xml (alias xml) (DEFAULT if Accept: / is sent)
  - text/turtle (returns text/n3)
  - text/n3
- DESCRIBE
  - application/rdf+xml (alias xml) (DEFAULT if Accept: / is sent)
  - text/turtle (returns text/n3)
  - text/n3

Valid alias for SELECT and ASK "xml", "json"

Valid alias for DESCRIBE and CONSTRUCT "xml", "json" (but it returns unexpected "application/sparql-results+json")

1.4.11 GraphDB

Website GraphDB, formerly known as OWLIM (OWLIM-Lite, OWLIM-SE)

Documentation https://graphdb.ontotext.com/documentation/free/

Uses Only content negotiation (no URL parameters).

Note If the Accept value is not within the expected ones, the server returns a 406 "No acceptable file format found."

JSON-LD (application/ld+json) supported (in CONSTRUCT and DESCRIBE).

- SELECT
  - application/sparql-results+xml, application/xml (.srx file)
  - application/sparql-results+json, application/json (.srj file)
  - text/csv (DEFAULT if Accept: / is sent)
  - text/tab-separated-values
- ASK
  - application/sparql-results+xml, application/xml (.srx file)
  - application/sparql-results+json (DEFAULT if Accept: / is sent), application/json (.srj file)
  - NOT supported: text/csv, text/tab-separated-values
- CONSTRUCT
  - application/rdf+xml, application/xml (.rdf file)
  - text/turtle (.ttl file)
  - application/n-triples (.nt file) (DEFAULT if Accept: / is sent)
  - text/n3, text/rdf+n3 (.n3 file)
  - application/ld+json (.jsonld file)
- DESCRIBE
  - application/rdf+xml, application/xml (.rdf file)
  - text/turtle (.ttl file)
  - application/n-triples (.nt file) (DEFAULT if Accept: / is sent)
  - text/n3, text/rdf+n3 (.n3 file)
  - application/ld+json (.jsonld file)

1.4.12 Stardog

Website Stardog

Documentation https://www.stardog.com/docs/#_http_headers_content_type_accept (looks outdated)

Uses Only content negotiation (no URL parameters).

Parameter key If an unexpected parameter is used, the server ignores it.

JSON-LD (application/ld+json) supported (in CONSTRUCT and DESCRIBE).

- SELECT
  - application/sparql-results+xml (DEFAULT if Accept: / is sent)
  - application/sparql-results+json
  - text/csv
  - text/tab-separated-values
  - Other values: application/x-binary-rdf-results-table
- ASK
  - application/sparql-results+xml (DEFAULT if Accept: / is sent)

  - application/sparql-results+json
  - Other values: text/boolean
  - Not supported: text/csv
  - Not supported: text/tab-separated-values
- CONSTRUCT
  - application/rdf+xml
  - text/turtle (DEFAULT if Accept: / is sent)
  - text/n3
  - application/ld+json
  - Other acceptable values: application/n-triples, application/x-turtle, application/trig, application/trix, application/n-quads
- DESCRIBE
  - application/rdf+xml
  - text/turtle (DEFAULT if Accept: / is sent)
  - text/n3
  - application/ld+json
  - Other acceptable values: application/n-triples, application/x-turtle, application/trig, application/trix, application/n-quads

1.5 Development

1.5.1 Requirements

The RDFLib package is used for RDF parsing.

This package is imported in a lazy fashion, i.e. only when needed. 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.

1.5.2 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 dev packages: `pip install '.[dev]'`
- `scripts`: some scripts to run the package against some SPARQL endpoints.
- `docs`: the documentation.

1.5.3 Community

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

1.5.4 Issues

Please, report any issue to github.

1.5.5 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).

1.5.6 License

The SPARQLWrapper package is licensed under W3C license.

1.5.7 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

TWO

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: str*, *updateEndpoint: Optional[str] = None*, *returnFormat: str = 'xml'*, *defaultGraph: Optional[str] = None*, *agent: str = 'sparqlwrapper 2.0.0 (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** – SPARQL endpoint's URI for SPARQL Update operations (if it's a different one).

The **default** value is `None`. :vartype updateEndpoint: string :ivar agent: The User-Agent for the HTTP request header. The **default** value is an autogenerated string using t he SPARQLWrapper version code. :vartype agent: string :ivar _defaultGraph: 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`. :vartype _defaultGraph: string :ivar user: The username of the credentials for querying the current endpoint. The value can be set an explicit call `setCredentials()`. The **default** value is `None`. :vartype user: string :ivar passwd: The password of the credentials for querying the current endpoint. The value can be set an explicit call `setCredentials()`. The **default** value is `None`. :vartype passwd: string :ivar http_auth: HTTP Authentication type. The **default** value is *`BASIC`*. Possible values are *`BASIC`* or *`DIGEST`*. It is used only in case the credentials are set. :vartype http_auth: string :ivar onlyConneg: Option for allowing (or not) **only** HTTP Content Negotiation (so dismiss the use of HTTP parameters). The default value is `False`. :vartype onlyConneg: boolean :ivar customHttpHeaders: 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). :vartype customHttpHeaders: dict :ivar timeout: The timeout (in seconds) to use for querying the endpoint. :vartype timeout: int :ivar queryString: The SPARQL query text. :vartype queryString: string :ivar queryType: 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). :vartype queryType: string :ivar 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`*. :vartype returnFormat: string :ivar requestMethod: The request method for query or update operations. The possibles values are URL-encoded (*`URLENCODED`*) or POST directly (*`POSTDIRECTLY`*). :vartype requestMethod: string :ivar method: The invocation method (HTTP verb). The **default** value is *`GET`*, but it can be set to *`POST`*. :vartype method: string :ivar parameters: The parameters of the request (key/value pairs in a dictionary). :vartype parameters: dict :ivar _defaultReturnFormat: The default return format. It is used in case the same class instance is reused for subsequent queries. :vartype _defaultReturnFormat: string

Variables

- **`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`** – 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`*. :vartype pattern: `re.RegexObject`, a compiled regular expression. See the `re` module of Python :cvar comments_pattern: regular expression used to remove comments from a query. :vartype comments_pattern: `re.RegexObject`, a compiled regular expression. See the `re` module of Python

`__init__`(*endpoint: str, updateEndpoint: Optional[str] = None, returnFormat: str = 'xml', defaultGraph: Optional[str] = None, agent: str = 'sparqlwrapper 2.0.0 (rdflib.github.io/sparqlwrapper)'*) → None

Class encapsulating a full SPARQL call.

Parameters

- **`endpoint`** (*`string`*) – SPARQL endpoint's URI.
- **`updateEndpoint`** – SPARQL endpoint's URI for update operations (if it's a different one). The **default**

value is `None`. :type updateEndpoint: string :param 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`*. :param defaultGraph: 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`. :type defaultGraph: string :param agent: The User-Agent for the HTTP request header. The **default** value is an autogenerated string using the SPARQLWrapper version number. :type agent: string

`resetQuery`() → None

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: str*) → None
Set the return format. If the one set is not an allowed value, the setting is ignored.

Parameters **`format`** – Possible values are *`JSON`*, *`XML`*, *`TURTLE`*, *`N3`*, *`RDF`*,

`RDFXML`, *`CSV`*, *`TSV`*, *`JSONLD`* (constants in this module). All other cases are ignored. :type format: string :raises ValueError: If *`JSONLD`* is tried to set and the current instance does not support `JSON-LD`.

`supportsReturnFormat`(*format: str*) → bool
Check if a return format is supported.

Parameters **`format`** – Possible values are *`JSON`*, *`XML`*, *`TURTLE`*, *`N3`*, *`RDF`*,

`RDFXML`, *`CSV`*, *`TSV`*, *`JSONLD`* (constants in this module). All other cases are ignored. :type format: string :return: Returns `True` if the return format is supported, otherwise `False`. :rtype: bool

`setTimeout`(*timeout: int*) → None
Set the timeout (in seconds) to use for querying the endpoint.

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

`setOnlyConneg`(*onlyConneg: bool*) → None
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`** – `True` if **only** HTTP Content Negotiation is allowed; `False` if HTTP parameters

are used. :type onlyConneg: bool

`setRequestMethod`(*method: str*) → None
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`** – Possible values are *`URLENCODED`* (URL-encoded) or *`POSTDIRECTLY`* (POST directly).

All other cases are ignored. :type method: string

`addDefaultGraph`(*uri: str*) → None
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: str*) → None
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: str, value: str*) → None
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: str*, *value: str*) → bool

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: str*, *value: str*) → bool

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: str*, *httpHeaderValue: str*) → None

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: str*) → bool

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: str*) → bool

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: Optional[str], passwd: Optional[str], realm: str = 'SPARQL'*) → None

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: str*) → None

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: Union[str, bytes]*) → None

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: str*) → Optional[str]

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: str*) → None

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() → None
Make `urllib2` use keep-alive.

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

isSparqlUpdateRequest() → bool
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() → bool
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: str*) → str
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: Optional[Tuple[str, str]] = None*) → str
Internal method for getting the request encoded parameters.

Parameters query – 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. :type query: tuple :return: the request encoded parameters. :rtype: string

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

See also:

`Hypertext Transfer Protocol – HTTP/1.1 - Header Field Definitions

<https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.1>`_

_createRequest() → urllib.request.Request
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*. :return: request a `urllib2.Request` object of the `urllib2` Python library

_query() → Tuple[http.client.HTTPResponse, str]
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() → *SPARQLWrapper.Wrapper.QueryResult*

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() → Optional[Union[bytes, str, Dict[Any, Any], Graph, xml.dom.minidom.Document]]

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: Union[http.client.HTTPResponse, Tuple[http.client.HTTPResponse, str]]*)

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*. :type requestedFormat: string

__init__(*result: Union[http.client.HTTPResponse, Tuple[http.client.HTTPResponse, str]]*) → None

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

format: (response, format).

`geturl()` → str
Return the URL of the original call.

Returns URL of the original call.

Return type string

`info()` → *SPARQLWrapper.KeyCaseInsensitiveDict.KeyCaseInsensitiveDict*[str]
Return the meta-information of the HTTP result.

Returns meta-information of the HTTP result.

Return type dict

`_convertJSON()` → Dict[Any, Any]
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()` → xml.dom.minidom.Document
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()` → Graph
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()` → bytes
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()` → bytes
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()` → bytes
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()` → Graph
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``()` → Optional[Union[bytes, str, Dict[Any, Any], Graph, xml.dom.minidom.Document]]

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``()` → Optional[str]

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`*. :rtype: string

`print_results``(`*minWidth: Optional[int] = None*`)` → None

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

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

2.2 SPARQLWrapper.SmartWrapper module

class `SPARQLWrapper.SmartWrapper.`**`Bindings`**`(`*retval:* SPARQLWrapper.Wrapper.QueryResult`)`

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: SPARQLWrapper.Wrapper.QueryResult*)

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

`convert`() → *SPARQLWrapper.SmartWrapper.Bindings*

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: str*) → Optional[List[*SPARQLWrapper.SmartWrapper.Value*]]

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: str, defaultGraph: Optional[str] = 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: str, defaultGraph: Optional[str] = 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`() → Union[*SPARQLWrapper.SmartWrapper.Bindings*, *SPARQLWrapper.Wrapper.QueryResult*]

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`() → Optional[Union[*SPARQLWrapper.SmartWrapper.Bindings*, *SPARQLWrapper.Wrapper.QueryResult*, bytes, str, Dict[Any, Any], Graph, xml.dom.minidom.Document]]

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: Optional[str]*) → None

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: str*, *binding: Dict[str, str]*)

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: str*, *binding: Dict[str, str]*) → None

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: Optional[bytes] = None*)

Bases: `Exception`

Base class for SPARQL Wrapper exceptions

__init__(*response: Optional[bytes] = None*)

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

exception SPARQLWrapper.SPARQLExceptions.**EndPointInternalError**(*response: Optional[bytes] = None*)

Bases: *SPARQLWrapper.SPARQLExceptions.SPARQLWrapperException*

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

exception SPARQLWrapper.SPARQLExceptions.**QueryBadFormed**(*response: Optional[bytes] = None*)

Bases: *SPARQLWrapper.SPARQLExceptions.SPARQLWrapperException*

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

exception SPARQLWrapper.SPARQLExceptions.**EndPointNotFound**(*response: Optional[bytes] = None*)

Bases: *SPARQLWrapper.SPARQLExceptions.SPARQLWrapperException*

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

exception SPARQLWrapper.SPARQLExceptions.**Unauthorized**(*response: Optional[bytes] = 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: Optional[bytes] = 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. ..

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 * Foundation CTIC :license: W3C® Software notice and license

class SPARQLWrapper.KeyCaseInsensitiveDict.**KeyCaseInsensitiveDict**(*d: Mapping[str, SPARQLWrapper.KeyCaseInsensitiveDict._V] = {}*)

Bases: `Dict[str, SPARQLWrapper.KeyCaseInsensitiveDict._V]`

A simple implementation of a key case-insensitive dictionary

`__init__`(*d: Mapping[str, SPARQLWrapper.KeyCaseInsensitiveDict._V] = {}*) → None

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

CHAPTER

THREE

SPARQLWRAPPER'S CHANGELOG

3.1 2022-03-14 2.0.0

- Ported codebase to Python 3. Dropped support for Python 2
- Removed nosetest in favour of unittest
- Added a CLI: `rqw`
- Updated for RDFLib >= 6.1.1
- Added type hints

Special thanks to @eggplants for making this release happen.

3.2 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

3.3 2019-04-18 1.8.4

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

3.4 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)

3.5 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)

3.6 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)

3.7 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)

3.8 2015-12-18 1.7.6

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

3.9 2015-11-19 1.7.5

- Removed pip dependency on setup (issue #69)

3.10 2015-11-05 1.7.4

- Fixed packaging (issue #66)

3.11 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

3.12 2015-11-03 1.7.2

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

3.13 2015-10-29 1.7.1

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

3.14 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

3.15 2014-08-26 1.6.4

- Fixed unicode problems on setup (issue #42)

3.16 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)

3.17 2014-07-24 1.6.2

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

3.18 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)

3.19 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

3.20 2012-08-28 1.5.2

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

3.21 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

3.22 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

3.23 2011-01-28 1.4.2

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

3.24 2010-01-11 1.4.1

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

3.25 2009-12-14 1.4.0

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

3.26 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)

3.27 2009-09-11 1.3.1

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

3.28 2009-05-06 1.3.0

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

3.29 2009-04-27 1.2.1

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

3.30 2008-07-10 1.2.0

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

3.31 2008-03-24 1.1.0

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

3.32 2008-03-07 1.0.1

- Fixed some cosmetic things.

3.33 2008-02-14 1.0.0

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

3.34 2007-07-06 0.2.0

- First public release of the library.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX