Package 'rdflib'

March 9, 2018

Title Tools to Manipulate and Query Semantic Data

Version 0.1.0

Description The Resource Description Framework, or 'RDF' is a widely used data representation model that forms the cornerstone of the Semantic Web. 'RDF' represents data as a graph rather than the familiar data table or rectangle of relational databases. The 'rdflib' package provides a friendly and concise user interface for performing common tasks on 'RDF' data, such as reading, writing and converting between the various serializations of 'RDF' data, including 'rdfxml', 'turtle', 'nquads', 'ntriples', and 'json-ld'; creating new 'RDF' graphs, and performing graph queries using 'SPARQL'. This package wraps the low level 'redland' R package which provides direct bindings to the 'redland' C library. Additionally, the package supports the newer and more developer friendly 'JSON-LD' format through the 'jsonld' package. The package interface takes inspiration from the Python 'rdflib' library.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL https://github.com/ropensci/rdflib

BugReports https://github.com/ropensci/rdflib/issues

Imports redland, jsonld, methods, utils, stringi, readr

RoxygenNote 6.0.1

Suggests magrittr, covr, testthat, knitr, rmarkdown, jqr, DT, tidyverse, dplyr, tidyr, tibble, purrr, lubridate, httr, xml2, jsonlite, repurrrsive, nycflights13, codemetar

VignetteBuilder knitr

NeedsCompilation no

Author Carl Boettiger [aut, cre, cph] (<https://orcid.org/0000-0002-1642-628X>)

Maintainer Carl Boettiger <cboettig@gmail.com>

Repository CRAN
Date/Publication 2018-03-09 12:56:58 UTC

R topics documented:

`rdflib-package` *rdflib: Tools to Manipulate and Query Semantic Data*

Description

The Resource Description Framework, or RDF is a widely used data representation model that forms the cornerstone of the Semantic Web. 'RDF' represents data as a graph rather than the familiar data table or rectangle of relational databases.

Details

It has three main goals:

- Easily read, write, and convert between all major RDF serialization formats
- Support SPARQL queries to extract data from an RDF graph into a data.frame
- Support JSON-LD format as a first-class citizen in RDF manipulations

For more information, see the Wikipedia pages for RDF, SPARQL, and JSON-LD:

- https://en.wikipedia.org/wiki/Resource_Description_Framework
- https://en.wikipedia.org/wiki/SPARQL
- https://en.wikipedia.org/wiki/JSON-LD

To learn more about rdflib, start with the vignettes: `browseVignettes(package = "rdflib")`

Configurations via `options()`

`rdflib_print_format`:

- NULL or "nquads" (default)
- any valid serializer name: e.g. "rdfxml", "jsonld", "turtle", "ntriples"

`rdflib_base_uri`:

- Default base URI to use (when serializing JSON-LD only at this time) default is "localhost://"

Author(s)

Maintainer: Carl Boettiger <cboettig@gmail.com> (0000-0002-1642-628X) [copyright holder]

See Also

Useful links:

- https://github.com/ropensci/rdflib
- Report bugs at https://github.com/ropensci/rdflib/issues

---

c.rdf *Concatenate rdf Objects.*

Description

All subsequent rdf objects will be appended to the first rdf object Note: this does not free memory from any of the individual rdf objects

Usage

```
## S3 method for class 'rdf'
c(...)
```

Arguments

...	objects to be concatenated

---

rdf *Initialize an* rdf *Object*

Description

Initialize an rdf Object

Usage

```
rdf(storage = c("memory", "BDB"), path = ".", new_db = FALSE)
```

Arguments

storage	Use in-memory hashes ("memory"), or disk based storage ("BDB")?
path	where should local database to store RDF triples be created, if configured for disk-based storage; see details.
new_db	logical, default FALSE. should we create a new database on disk or attempt to connect to an existing database (at the path specified)?

Details

an rdf Object is a list of class 'rdf', consisting of three pointers to external C objects managed by the redland library. These are the `world` object: basically a top-level pointer for all RDF models, and a `model` object: a collection of RDF statements, and a `storage` object, indicating how these statements are stored.

`rdflib` defaults to an in-memory hash-based storage structure. which should be best for most use cases. For very large triplestores, disk-based storage will be necessary. set `storage="BDB"` to use disk-based storage. Specify a path with the optional `path` argument, default uses the current working directory. Disk-based storage requires redland package to be installed from source with support for the Berkeley DB (libdb-dev on Ubuntu, berkeley-db on homebrew), otherwise this will fall back to in-memory storage with a warning. Check for working BDB support with the function `rdf_has_bdb()`.

Typical use will be simply to initialize a container to which the user would manually add triples using `rdf_add`.

Value

an rdf object

Examples

```
x <- rdf()
```

`rdf_add` *Add RDF Triples*

Description

add a triple (subject, predicate, object) to the RDF graph

Usage

```
rdf_add(rdf, subject, predicate, object, subjectType = as.character(NA),
  objectType = as.character(NA), datatype_uri = as.character(NA))
```

Arguments

`rdf`	an rdf object
`subject`	character string containing the subject
`predicate`	character string containing the predicate
`object`	character string containing the object
`subjectType`	the Node type of the subject, i.e. "uri", "blank"
`objectType`	the Node type of the object, i.e. "literal", "uri", "blank"
`datatype_uri`	the datatype URI to associate with a object literal value

Details

`rdf_add()` will automatically 'duck type' nodes (if looks like a duck...). That is, strings that look like URIs will be declared as URIs. (See URI). Predicate should always be a URI (e.g. URL or a `prefix:string`), cannot be blank or literal. Subjects that look like strings will be treated as Blank Nodes (i.e. will be prefixed with `_:`). An empty subject, `""`, will create a blank node with random name. Objects that look like URIs will be typed as resource nodes, otherwise as literals. An empty object `""` will be treated as blank node. Set `subjectType` or `objectType` explicitly to override this behavior, e.g. to treat an object URI as a literal string. NAs are also treated as blank nodes in subject or object See examples for details.

Value

Silently returns the updated RDF graph (rdf object). Since the rdf object simply contains external pointers to the model object in C code, note that the input object is modified directly, so you need not assign the output of rdf_add() to anything.

References

https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

Examples

```
rdf <- rdf()
rdf_add(rdf,
    subject="http://www.dajobe.org/",
    predicate="http://purl.org/dc/elements/1.1/language",
    object="en")

## non-URI string in subject indicates a blank subject
## (prefixes to "_:b0")
rdf_add(rdf, "b0", "http://schema.org/jobTitle", "Professor")

## identically a blank subject.
## Note rdf is unchanged when we add the same triple twice.
rdf_add(rdf, "b0", "http://schema.org/jobTitle", "Professor",
        subjectType = "blank")

## blank node with empty string creates a default blank node id
rdf_add(rdf, "", "http://schema.org/jobTitle", "Professor")


## Subject and Object both recognized as URI resources:
rdf_add(rdf,
        "https://orcid.org/0000-0002-1642-628X",
        "http://schema.org/homepage",
        "http://carlboettiger.info")

 ## Force object to be literal, not URI resource
rdf_add(rdf,
        "https://orcid.org/0000-0002-1642-628X",
        "http://schema.org/homepage",
```

```
"http://carlboettiger.info",
objectType = "literal")
```

---

rdf_free *Free Memory Associated with RDF object*

Description

Free Memory Associated with RDF object

Usage

```
rdf_free(rdf, rm = TRUE)
```

Arguments

rdf	an rdf object
rm	logical, default TRUE. Remove pointer from parent.frame()? Usually a good idea since referring to a pointer after it has been removed can crash R.

Details

Free all pointers associated with an rdf object. Frees memory associated with the storage, world, and model objects.

Examples

```
rdf <- rdf()
rdf_free(rdf)
rm(rdf)
```

---

rdf_has_bdb *Check for BDB support*

Description

Detect whether Berkeley Database for disk-based storage of RDF graphs is available. Disk-based storage requires redland package to be installed from source with support for the Berkeley DB (libdb-dev on Ubuntu, berkeley-db on homebrew), otherwise `rdf()` will fall back to in-memory storage with a warning.

Usage

```
rdf_has_bdb()
```

Value

TRUE if BDB support is detected, false otherwise

Examples

```
rdf_has_bdb()
```

`rdf_parse` *Parse RDF Files*

Description

Parse RDF Files

Usage

```
rdf_parse(doc, format = c("guess", "rdfxml", "nquads", "ntriples", "turtle",
  "jsonld"), rdf = NULL, base = getOption("rdflib_base_uri",
  "localhost://"), ...)
```

Arguments

`doc`	path, URL, or literal string of the rdf document to parse
`format`	rdf serialization format of the doc, one of "rdfxml", "nquads", "ntriples", "turtle" or "jsonld". If not provided, will try to guess based on file extension and fall back on rdfxml.
`rdf`	an existing rdf triplestore to extend with triples from the parsed file. Default will create a new rdf object.
`base`	the base URI to assume for any relative URIs (blank nodes)
`...`	additional parameters (not implemented)

Value

an rdf object, containing the redland world and model objects

Examples

```
doc <- system.file("extdata", "dc.rdf", package="redland")
rdf <- rdf_parse(doc)
```

rdf_query *Perform a SPARQL Query*

Description

Perform a SPARQL Query

Usage

```
rdf_query(rdf, query, data.frame = TRUE, ...)
```

Arguments

rdf	an rdf object (e.g. from rdf_parse)
query	a SPARQL query, as text string
data.frame	logical, should the results be returned as a data.frame?
...	additional arguments to a redland initialize-Query

Value

a data.frame of all query results (default.) Columns will be named according to variable names in the SPARQL query. Returned object values will be coerced to match the corresponding R type to any associated datatype URI, if provided. If a column would result in mixed classes (e.g. strings and numerics), all types in the column will be coerced to character strings. If data.frame is false, results will be returned as a list with each element typed by its data URI.

Examples

```
doc <- system.file("extdata", "dc.rdf", package="redland")

sparql <-
'PREFIX dc: <http://purl.org/dc/elements/1.1/>
 SELECT ?a ?c
 WHERE { ?a dc:creator ?c . }'

rdf <- rdf_parse(doc)
rdf_query(rdf, sparql)
```

rdf_serialize *Serialize an RDF Document*

Description

Serialize an RDF Document

Usage

```
rdf_serialize(rdf, doc = NULL, format = c("guess", "rdfxml", "nquads",
  "ntriples", "turtle", "jsonld"), namespace = NULL, prefix = NULL,
  base = getOption("rdflib_base_uri", "localhost://"), ...)
```

Arguments

rdf	an existing rdf triplestore to extend with triples from the parsed file. Default will create a new rdf object.
doc	file path to write out to. If null, will write to character.
format	rdf serialization format of the doc, one of "rdfxml", "nquads", "ntriples", "turtle" or "jsonld". If not provided, will try to guess based on file extension and fall back on rdfxml.
namespace	string giving the namespace to set
prefix	string giving the prefix associated with the namespace
base	the base URI to assume for any relative URIs (blank nodes)
...	additional arguments to `redland::serializeToFile`

Value

rdf_serialize returns the output file path doc invisibly. This makes it easier to use rdf_serialize in pipe chains with rdf_parse.

Examples

```
infile <- system.file("extdata", "dc.rdf", package="redland")
out <- tempfile("file", fileext = ".rdf")

rdf <- rdf_parse(infile)
rdf_serialize(rdf, out)

## With a namespace
rdf_serialize(rdf,
          out,
          namespace = "http://purl.org/dc/elements/1.1/",
          prefix = "dc")
```

Index