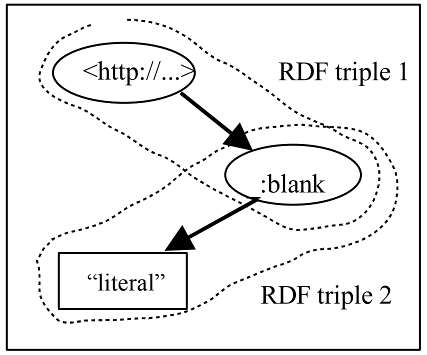
SPARQL RDF Query Language Reference v1.7

Copyright © 2005 Dave Beckett, ILRT, University of Bristol.
Latest version: <http://www.ilrt.bris.ac.uk/people/cmdjb/2005/04-sparql/>
Comments to: dave.beckett@bristol.ac.uk

1. RDF Model and SPARQL RDF Terms Syntax

RDF Graph:	A set of RDF Triples
RDF Triple:	A triple (3-tuple) of:
Subject:	URI or Blank Node
Predicate:	URI
Object:	URI or Blank Node or Literal

RDF Graph



URI: An absolute URI which may include a # fragment.
`<http://www.w3.org/>`
`<http://example.org/#fragment>`

`<abc.rdf>`	Relative URI resolved against base URI.
`<>`	Base URI, usually the query document URI
`ex:name`	URI shorthand using XML-style prefix `ex` and local name. Declared with `PREFIX` (SPARQL) or `@prefix` (Turtle)

RDF Literal: A Unicode string with an optional language tag.
`“hello”      “bonjour”@fr`

RDF Typed Literal: A Unicode string and datatype URI for encoding datatypes.
`“abc”^^<http://example.org/myDatatype>`
abbreviated with an XML QName style as:
`“10”^^xsd:integer`
Short forms for several common datatypes:

`10`	`“10”^^xsd:integer`
`1.2e3`	`“1.2e3”^^xsd:double`
`true`	`“true”^^xsd:boolean`

Blank Node: A node in a graph with a local name. The scope of the name is the RDF graph.
`_:node`

2. Common RDF Namespaces and Prefixes

Namespace	Common Prefix	Namespace URI
RDF	rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
Dublin Core	dc:	http://purl.org/dc/elements/1.1/
FOAF	foaf:	http://xmlns.com/foaf/0.1/
XML Schema Datatypes	xsd:	http://www.w3.org/2001/XMLSchema#
RDFS	rdfs:	http://www.w3.org/2000/01/rdf-schema#
OWL	owl:	http://www.w3.org/2002/07/owl#

3. SPARQL Query Language Reference

Based on SPARQL Query 21 July 2005 <http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050721/>.

RDF Term:	A part of an RDF Triple. A URI, Blank Node or a Literal. `<uri> _:b1 “Literal”@en “abc123”^^my:datatype`
Query Variable:	Identifiers for binding to RDF Terms in matches. `?a / $b` or in lists: `$name $title $place`
Anonymous Query Variable:	Blank Nodes in a graph pattern act as variables that cannot be `SELECT`ed `_:abc`

Triple Pattern:	An RDF Triple with Query Variables or blank nodes allowed in each term:
	`<http://example.org/abc> ?x “Hello”`
	`?subject ?predicate ?object`
	Turtle abbreviations can be used for Triple Patterns, see **Section 4**.

Graph Pattern:	A block that matches part of the queried RDF graph.

Basic Graph Pattern:	A set of Triple Patterns binding RDF Terms in the graph to variables.
	Written as a `{..}` block with `'.'` separating the triple patterns:
	`{ <http://example.org/abc> ?y “Hello” .`
	` ?subject $predicate “Literal” }`

Group Graph Pattern:	A graph pattern containing multiple graph patterns which must all match
	`{ { ?person rdf:type  foaf:Person }`
	`  { ?person foaf:name “Dave” } }`

Optional Graph Pattern:	A graph pattern which may fail to match and provide bindings but not cause the entire query to fail. Written with `OPTIONAL` before a graph pattern. `OPTIONAL { ?person foaf:nick ?nick }`

Union Graph Pattern:	A pair of graph patterns any of which may match and bind the same variables. Written with the `UNION` keyword between two graph patterns.
	`{ ?node ex:name ?name } UNION`
	`{ ?node vcard:FN ?name }`

Graph Graph Pattern:	A keyword for specifying a graph name to use or to return a graph name as a binding. Written with the `GRAPH` keyword before a graph pattern.
	`GRAPH <http://example.org/myfoaf>`
	`  { ?person foaf:name ?name }`
	`GRAPH ?graph { ?person foaf:name ?name }`

Value Constraints:	A boolean expression in a graph pattern over query variables that constrains matched graph patterns. `{ ?item ex:size $size .  FILTER ($size < 10 ) }`

4. SPARQL Query Language Structure

Prologue (optional)	`BASE` *<uri>* `PREFIX` *prefix*`:` *<uri>* (repeatable)

Query Result forms (required, pick 1)	`SELECT (DISTINCT)` sequence of *?variable*
	`SELECT (DISTINCT)*`
	`DESCRIBE` sequence of *?variable* or *<uri>*
	`DESCRIBE *`
	`CONSTRUCT {` *graph pattern* `}`
	`ASK`

Query Dataset Sources (optional)	Add triples to the background graph (repeatable): `FROM` *<uri>* Add a named graph (repeatable): `FROM NAMED` *<uri>*
Graph Pattern (optional, required for `ASK`**)**	`WHERE {` *graph pattern* [`FILTER` *expression*]}
Query Results Ordering (optional)	`ORDER BY ...`
Query Results Selection (optional)	`LIMIT` *n*, `OFFSET` *m*

5. SPARQL Query Result Forms

Variable Bindings: A sequence of (set of variable bindings) for each query pattern match.

```
SELECT *
WHERE {$a rdf:type $b}
```

to ask for bindings for all variables mentioned in the query and

```
SELECT $a ?b
WHERE {$a rdf:type ?b}
```

to list them explicitly

RDF Graph:

Describe Resources: An RDF graph describing resources either given by URI

```
DESCRIBE <http://example.org/thing>
```

or by binding variables using the same syntax as `SELECT`.

```
DESCRIBE ?person
WHERE  { ?person foaf:name “Dave” }
```

Build an RDF graph An RDF graph made by substituting variables into a triple template.

```
CONSTRUCT { ?a foaf:knows ?b }
WHERE { ?a ex:KnowsQuiteWell ?b }
```

Boolean: True if the query pattern could be answered.

```
ASK
WHERE { ?a rdf:type foaf:Person }
```

6. Query Results Ordering and Modifying

The optional modifications on query results are performed in the following order:

1. `DISTINCT` to ensure solutions in the sequence are unique
1. `ORDER BY` ordering solutions sequences by variable, expression or extension function call:
   `ORDER BY DESC(?date) ?title ASC(?familyName) my:fn(?a)`
   in descending order by *date*, by ascending *title* order, by *familyName* ascending, by extension function
2. `LIMIT` *n* to restrict the number of solutions to *n*
3. `OFFSET` *m* to start the results in the solution from item *m*

7. Values – datatypes, expressions and operators

Supported datatypes: RDF Terms, `xsd:boolean, xsd:string, xsd:double, xsd:float, xsd:decimal, xsd:integer` and `xsd:dateTime`

Logical operators:	Logical:	*A* `||` *B*, *A* && *B*, `!`A, (*A*)
	Comparison (*A op B*):	=, !=, <, >, <=, >=
Arithmetic operators:	Unary:	+*A*, -*A*
	Binary (*A op B*):	+, -, *, /
RDF operators:	Boolean:	`BOUND`(*A*), `isURI`(*A*), `isBLANK`(*A*), `isLITERAL`(*A*)
	String:	`STR`(A), `LANG`(*A*), `DATATYPE`(*A*)
String Match operator:		`REGEX`(*string expression*, *pattern expression* [,*flags expression*] `)` *pattern* syntax is from XQuery 1.0 / XPath 2.0, XML Schema and similar to Perl. *flags* are s, m, i, x
Extension Functions and Explicit Type Casting:		*QName*`(` *expression*, *expression*, ...`)`
Automatic Type Promotion:	from `xsd:decimal`	to `xsd:float`
	from `xsd:float`	to `xsd:double`

8. Turtle RDF Syntax Reference

Turtle (Terse RDF Triple Language) describes triples in an RDF graph and allows abbreviations. Triple Patterns in SPARQL can use the same abbreviations.

This description is based on Turtle 2005-07-01 from <http://www.ilrt.bris.ac.uk/discovery/2004/01/turtle/>

RDF Terms:

URI	< *URI* >	(<> is the base URI, often the document URI)
Literal:	`"`*string*`"` or `"`*string*`"`@*language* or ^^< *datatype URI* >	
Blank Node:	`_:` *name* or `[]` for an anonymous blank node	

`@prefix` operator: URIs can be written as XML-style QNames by defining a prefix / URI binding:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
```

Triples: Written as 3 RDF terms with whitespace separating them as necessary, and '`.`' between triples:

```
<> dc:title "SPARQL Reference" .
<> dc:date “2005-04-19”^^xsd:dateTime .
```

`,` operator: Triples with the same subject and predicate may be abbreviated with '`,`':

```
<http://example.org/mybook> dc:title “My Book”, “Mein Buch”@de .
```

`;` operator: Triples with the same subject may be abbreviated with '`;`':

```
<http://work.example.org/> dc:title “My Workplace”;
                           dc:publisher “My Employer” .
```

`[ ... ]` operator: A sequence of (predicate object) pairs may be put inside `[` ... `]` and a blank node subject will be assigned to them:

```
<> dc:creator [ foaf:name “Dave”; foaf:homePage <http:...> ] .
```

`[]` operator: A blank node:

```
[] a ex:Book [ dc:title “Thing”; dc:description “On the shelf” ] .
```

`a` predicate: The often-used `rdf:type` QName may be abbreviated by the keyword `a` as a predicate:

```
<> a Foaf:Document .
```

Integers: Decimal integers 0 or larger can be written directly as literals (type `xsd:integer`)

```
<> ex:sizeInBytes 12345 .
```

`( ... )` collections: RDF collections can be written inside `(` ... `)` as a space-separated list of the contents:

```
<> ex:contents ( ex:apple ex:banana ex:pear ) .
```

9. Example SPARQL Query

```
BASE <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
# This is a relative URI to BASE above
PREFIX ex: <properties/1.0#>

SELECT DISTINCT $person ?name $age
FROM <http://rdf.example.org/personA.rdf>
FROM <http://rdf.example.org/personB.rdf>
WHERE {  $person a foaf:Person ;
                 foaf:name ?name.
         OPTIONAL { $person ex:age $age } .
         FILTER (!REGEX(?name, “Bob”))
       }
ORDER BY ASC(?name) LIMIT 10 OFFSET 20
```