SPARQL Update

A language for updating RDF graphs

W3C Member Submission 15 July 2008

This version:
http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/
Latest version:
http://www.w3.org/Submission/SPARQL-Update/
Authors:
Andy Seaborne, Hewlett-Packard
Geetha Manjunath, Hewlett-Packard
Chris Bizer, Freie Universität, Berlin
John Breslin, DERI Galway at the National University of Ireland, Galway
Souripriya Das, Oracle Corporation
Ian Davis, Talis Information Systems Ltd
Steve Harris, Garlik Ltd.
Kingsley Idehen, OpenLink Software Inc.
Olivier Corby, INRIA
Kjetil Kjernsmo, Computas AS
Benjamin Nowack, Semsol

Abstract

This document describes SPARQL/Update, an update language for RDF graphs. It uses a syntax derived form SPARQL. Update operations are performed on a collection of graphs in a Graph Store. Operations are provided to change existing RDF graphs as well as create and remove graphs with the Graph Store.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is a member submission, offered by Hewlett-Packard, on behalf of the submitting organizations above.

By publishing this document, W3C acknowledges that the Submitting Members have made a formal Submission request to W3C for discussion. Publication of this document by W3C indicates no endorsement of its content by W3C, nor that W3C has, is, or will be allocating any resources to the issues addressed by it. This document is not the product of a chartered W3C group, but is published as potential input to the W3C Process. A W3C Team Comment

---

Table of Contents

1 Introduction

SPARQL/Update is a language to express updates to an RDF store. It is intended to be a standard mechanism by which updates to a remote RDF Store can be described, communicated and stored. SPARQL/Update is a companion language to SPARQL and is envisaged to be used in conjunction with the SPARQL query language. The reuse of the SPARQL syntax, in both style and detail, reduces the learning curve for developers and reduces implementation costs.

SPARQL/Update provides the following facilities:

- Insert new triples to an RDF graph.
- Delete triples from an RDF graph.
- Perform a group of update operations as a single action.
- Create a new RDF Graph to a Graph Store.
- Delete an RDF graph from a Graph Store.

Related work:

- the wiki page SparqlUpdateLanguage for a related proposal
- Delta: an ontology for the distribution of differences between RDF graphs

1.1 Scope and Limitations

SPARQL/Update is not:

- a replacement for RSS or Atom, which may be used to advertise changes.
- a mechanism to exchange changes to RDF graphs between applications.
- a general web-wide approach to update and distribution of RDF-based information.

1.2 Document Conventions

Language forms are show as:

```
INSERT { template } [ WHERE { pattern } ]
```

`[]` indicates an optional part of the syntax. `[]` are still used for blank nodes in SPARQL requests.

Italics indicate an item is some syntax element derived from SPARQL.

Examples are shown as:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT { <http://example/egbook3> dc:title  "This is an example title" }
```

2 Examples

This section gives some example snippets in the proposed SPARQL/Update language that would be used to update a remote RDF store.

(a) Adding some triples to a graph. The snippet describes two RDF triples to be inserted into the default graph of the RDF store.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT DATA
{ <http://example/book3> dc:title    "A new book" ;
                         dc:creator  "A.N.Other" .
}
```

(b) This SPARQL/Update request contains a triple to be deleted and a triple to be added (used here to correct a book title). The requested change happens in the named graph identified by the URI `http://example/bookStore`.

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

DELETE DATA FROM <http://example/bookStore>
{ <http://example/book3>  dc:title  "Fundamentals of Compiler Desing" }

INSERT DATA INTO <http://example/bookStore>
{ <http://example/book3>  dc:title  "Fundamentals of Compiler Design" }
```

(c) The example below has a request to delete all records of old books (with date before year 2000)

```
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

DELETE
 { ?book ?p ?v }
WHERE
  { ?book dc:date ?date .
    FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
    ?book ?p ?v
  }
```

(d) This snippet copies records from one named graph to another named graph based on a pattern

```
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

INSERT INTO <http://example/bookStore2>
 { ?book ?p ?v }
WHERE
  { GRAPH  <http://example/bookStore>
       { ?book dc:date ?date .
         FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
       ?book ?p ?v
  } }
```

(e) An example to move records from one named graph to another named graph based on a pattern

```
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

INSERT INTO <http://example/bookStore2>
 { ?book ?p ?v }
WHERE
  { GRAPH  <http://example/bookStore>
     { ?book dc:date ?date .
       FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
       ?book ?p ?v
     }
  }

DELETE FROM <http://example/bookStore>
 { ?book ?p ?v }
WHERE
  { GRAPH  <http://example/bookStore>
      { ?book dc:date ?date .
        FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
        ?book ?p ?v
      }
  }
```

3 The Graph Store

A Graph Store is a repository of RDF graphs managed by a single service. Like an RDF Dataset operated on by SPARQL, a Graph Store has zero or one unnamed graphs and zero or more named graphs. Unlike an RDF dataset, named graphs can be added or deleted to a graph store.

In the simple case, where there is one default graph and no named graphs, SPARQL/Update is a language for update of a single graph.

3.1 Graph Store and SPARQL Query Services

If an update service is managing some graph store, then there is no presumption that this exactly corresponds to any RDF dataset offered by some query service. The dataset of the query service may be formed from graphs managed by the update service but the dataset offered by the query can be a subset of the graphs in the update service dataset and the names of those graphs may be different. The composition of the RDF dataset may also change.

3.2 SPARQL/Update Services

SPARQL/Update requests are a sequence of operations. Each request should be treated atomically by a SPARQL/Update service.

If two different update services are managing their respective graph stores share a common graph, there is no presumption that the updates done through one service will be propagated to the other. The behaviour of these services with respect to each other is implementation dependent. It is recommended that such deployment scenarios are avoided.

An implementation must target SPARQL queries on updated graphs if the SPARQL and SPARQL/Update end points are the same.

4 Update Language

SPARQL/Update supports two categories of update operations on a Graph Store:

- Graph Update - addition and removal of triples from one of the graphs in the graph store.
- Graph Management - creating and deletion of graphs within the graph store.

Multiple operations can be packed into requests. The operations of a request are executed in order given.

4.1 Graph Update

Graph update operations change existing graphs in the Graph Store but do not create or delete them.

The `INSERT DATA` operation adds some triples, which are inline in the request, into a graph.

The `DELETE DATA` operation removes some triples, which are inline in the request, from a graph

The fundamental pattern-based operation of a graph update is `MODIFY`. There are two restricted forms for `INSERT` and `DELETE` of triples by pattern; both are special cases of the `MODIFY` operation. A modify operation consists of a group of triples to be deleted and a group of triples to be added. The specification of the triples is based on a query pattern.

The difference between `MODIFY` / `INSERT` / `DELETE` and `INSERT DATA` / `DELETE DATA` is that `INSERT DATA` and `DELETE DATA` do not take a template and pattern. The `DATA` forms require concrete data (no named variables). Having specific operations means that a request can be streamed so that large, pure-data, updates can be done.

The `LOAD` operation reads the contents of a document representing a graph into a graph in the graph store.

The `CLEAR` operation removes all the triples in a graph.

4.1.1 INSERT DATA

Insert data into a graph:

```
INSERT DATA [ INTO <uri> ]*
{ triples }
```

4.1.2 DELETE DATA

Delete data from a graph:

```
DELETE DATA [ FROM <uri> ]*
{ triples }
```

4.1.3 MODIFY

```
# UPDATE outline syntax : general form:
MODIFY [ <uri> ]*
DELETE { template }
INSERT { template }
```

```
[ WHERE { pattern } ]
```

The `template` and `pattern` forms are as defined in SPARQL for construct templates and graph patterns.

The deletion of the triples happens before the insertion. The pattern in `WHERE` clause is evaluated only once, before the delete part of the operation is performed. The overall processing model is that the pattern is executed, the results used to instantiate the `DELETE` template, the deletes performed, the results used again to instantiate the `INSERT` template, and the inserts performed.

The `pattern` is evaluated as in a SPARQL query `SELECT * WHERE { pattern }` and all the named variables are made available to the `INSERT` template and the `DELETE` template for defining the triples to be inserted or deleted.

The `<uri>` clause names the graph in the graph store to be updated; if omitted, the `MODIFY` applies to the unnamed graph.

If the operation is on a graph that does not exist, an error is generated. Graphs are not created by the graph update operations. The graph management operations have to be used to create new graphs.

4.1.4 DELETE

The `DELETE` operation is similar to the `MODIFY` operation with an empty `INSERT` template. The graph URI, if present, must be a valid named graph in the Graph Store.

```
DELETE [ FROM <uri> ]* { template } [ WHERE { pattern } ]
```

```
# Equivalent to
MODIFY [ <uri> ]*
DELETE { template }
INSERT { }
[ WHERE { pattern } ]
```

4.1.5 INSERT

The `INSERT` operation is similar to the `MODIFY` operation with an empty `DELETE` template. The graph URI, if present, must be a valid named graph in the Graph Store. The `CREATE` graph management operator should be used to create a new named graph in the Graph Store.

```
INSERT [ INTO <uri> ]* { template } [ WHERE { pattern } ]
```

```
#Equivalent to
MODIFY [ GRAPH <uri> ]*
DELETE { }
INSERT { template }
[ WHERE { pattern } ]
```

4.1.6 LOAD

The `LOAD` operation copies all the triples of a remote graph into the specified graph, or if not specified, the default graph of the Graph Store.

```
LOAD <remoteURI> [ INTO <uri> ]
```

4.1.7 CLEAR

The CLEAR operations removes all the triples of the specified graph or if not specified, the default graph of the Graph Store. This operation does not remove the graph from the Graph Store.

```
# Remove all triples.
CLEAR [ GRAPH <uri> ]
```

It has the same effect as:

```
# Remove all triples.
DELETE [ <uri> ] { ?s ?p ?o } WHERE { ?s ?p ?o }
```

but is a clearer expression of empting a graph.

4.2 Graph Management

Graph management operations create and destroy named graphs in the Graph Store.

The default graph in a Graph Store always exists.

4.2.1 Graph Creation

```
CREATE [ SILENT ] GRAPH <uri>
```

This operation creates a new named graph with a name as specified by the URI. After the successful completion of this operation, the new named graph is available for any further graph update operations with MODIFY, INSERT and DELETE operators.

The SPARQL/Update service generates an error if the graph referred by the URI already exists unless the keyword SILENT is present when no error is generated and execution of the sequence of SPARQL/Update operations continues.

4.2.2 Graph Removal

```
DROP [ SILENT ] GRAPH <uri>
```

This operation removes the specified named graph from the Graph Store associated with the SPARQL/Update service endpoint. After successful completion of this operation, the named graph is no more available for further graph update operations.

The SPARQL/Update service, by default, is expected to generate an error if the specified named graph does not exist. If SILENT is present, this error is ignored and execution of a sequence of SPARQL/Update operations continues.

5 Security Issues

Exposing RDF data for update creates many security issues which any deployment must be aware of, and consider the risks involved. This submission does not describe such issues.

In addition to exposing update services over web technologies, implementers of both systems using SPARQL/Update and servers providing a graph store, must be aware that a string-based language is susceptible to insertion attacks in the construction of

SPARQL/Update requests.

6 SPARQL/Update Grammar

Rules from rule 16, `Prologue`, are taken from the SPARQL grammar.

> Note: This grammar does not use special rules for parsing INSERT DATA/DELETE DATA where only triples, not triple patterns (allowing named variables) are allowed. It assumes the update processor will check template contains only ground triples.

[1]	*SPARQLUpdate*	::=	Prologue ((Update | Manage))*
[2]	*Update*	::=	Modify | Insert | Delete | Load | Clear
[3]	*Modify*	::=	'MODIFY' GraphIRI* 'DELETE' ConstructTemplate 'INSERT' ConstructTemplate WhereClause?
[4]	*Delete*	::=	'DELETE' (DeleteData | DeleteTemplate)
[5]	*DeleteData*	::=	'DATA' ('FROM'? IRIref)* ConstructTemplate
[6]	*DeleteTemplate*	::=	('FROM'? IRIref)* ConstructTemplate WhereClause?
[7]	*Insert*	::=	'INSERT' (InsertData | InsertTemplate)
[8]	*InsertData*	::=	'DATA' ('INTO'? IRIref)* ConstructTemplate
[9]	*InsertTemplate*	::=	('INTO'? IRIref)* ConstructTemplate WhereClause?
[10]	*GraphIRI*	::=	'GRAPH' IRIref
[11]	*Load*	::=	'LOAD' IRIref+ ('INTO' IRIref)?
[12]	*Clear*	::=	'CLEAR' GraphIRI?
[13]	*Manage*	::=	Create | Drop
[14]	*Create*	::=	'CREATE' 'SILENT'? GraphIRI
[15]	*Drop*	::=	'DROP' 'SILENT'? GraphIRI
[16]	*Prologue*	::=	BaseDecl? PrefixDecl*
[17]	*BaseDecl*	::=	'BASE' IRI_REF
[18]	*PrefixDecl*	::=	'PREFIX' PNAME_NS IRI_REF
[19]	*WhereClause*	::=	'WHERE'? GroupGraphPattern
[20]	*GroupGraphPattern*	::=	'{' TriplesBlock? ((GraphPatternNotTriples | Filter) '.'? TriplesBlock?)* '}'
[21]	*TriplesBlock*	::=	TriplesSameSubject ('.' TriplesBlock?)?
[22]	*GraphPatternNotTriples*	::=	OptionalGraphPattern | GroupOrUnionGraphPattern | GraphGraphPattern
[23]	*OptionalGraphPattern*	::=	'OPTIONAL' GroupGraphPattern
[24]	*GraphGraphPattern*	::=	'GRAPH' VarOrIRIref GroupGraphPattern
[25]	*GroupOrUnionGraphPattern*	::=	GroupGraphPattern ('UNION' GroupGraphPattern)*
[26]	*Filter*	::=	'FILTER' Constraint
[27]	*Constraint*	::=	BrackettedExpression | BuiltInCall | FunctionCall
[28]	*FunctionCall*	::=	IRIref ArgList
[29]	*ArgList*	::=	(NIL | '(' Expression (',' Expression)* ')')
[30]	*ConstructTemplate*	::=	'{' ConstructTriples? '}'
[31]	*ConstructTriples*	::=	TriplesSameSubject ('.' ConstructTriples?)?
[32]	*TriplesSameSubject*	::=	VarOrTerm PropertyListNotEmpty | TriplesNode PropertyList
[33]	*PropertyListNotEmpty*	::=	Verb ObjectList (';' (Verb ObjectList)?)*
[34]	*PropertyList*	::=	PropertyListNotEmpty?
[35]	*ObjectList*	::=	Object (',' Object)*
[36]	*Object*	::=	GraphNode

[37]	*Verb*	::=	VarOrIRIref | 'a'
[38]	*TriplesNode*	::=	Collection | BlankNodePropertyList
[39]	*BlankNodePropertyList*	::=	'[' PropertyListNotEmpty ']'
[40]	*Collection*	::=	'(' GraphNode+ ')'
[41]	*GraphNode*	::=	VarOrTerm | TriplesNode
[42]	*VarOrTerm*	::=	Var | GraphTerm
[43]	*VarOrIRIref*	::=	Var | IRIref
[44]	*Var*	::=	VAR1 | VAR2
[45]	*GraphTerm*	::=	IRIref | RDFLiteral | NumericLiteral | BooleanLiteral | BlankNode | NIL
[46]	*Expression*	::=	ConditionalOrExpression
[47]	*ConditionalOrExpression*	::=	ConditionalAndExpression ('||' ConditionalAndExpression)*
[48]	*ConditionalAndExpression*	::=	ValueLogical ('&&' ValueLogical)*
[49]	*ValueLogical*	::=	RelationalExpression
[50]	*RelationalExpression*	::=	NumericExpression ('=' NumericExpression | '!=' NumericExpression | '<' NumericExpression | '>' NumericExpression | '<=' NumericExpression | '>=' NumericExpression)?
[51]	*NumericExpression*	::=	AdditiveExpression
[52]	*AdditiveExpression*	::=	MultiplicativeExpression ('+' MultiplicativeExpression | '-' MultiplicativeExpression | NumericLiteralPositive | NumericLiteralNegative)*
[53]	*MultiplicativeExpression*	::=	UnaryExpression ('*' UnaryExpression | '/' UnaryExpression)*
[54]	*UnaryExpression*	::=	'!' PrimaryExpression | '+' PrimaryExpression | '-' PrimaryExpression | PrimaryExpression
[55]	*PrimaryExpression*	::=	BrackettedExpression | BuiltInCall | IRIrefOrFunction | RDFLiteral | NumericLiteral | BooleanLiteral | Var
[56]	*BrackettedExpression*	::=	'(' Expression ')'
[57]	*BuiltInCall*	::=	'STR' '(' Expression ')' | 'LANG' '(' Expression ')' | 'LANGMATCHES' '(' Expression ',' Expression ')' | 'DATATYPE' '(' Expression ')' | 'BOUND' '(' Var ')' | 'sameTerm' '(' Expression ',' Expression ')' | 'isIRI' '(' Expression ')' | 'isURI' '(' Expression ')' | 'isBLANK' '(' Expression ')' | 'isLITERAL' '(' Expression ')' | RegexExpression
[58]	*RegexExpression*	::=	'REGEX' '(' Expression ',' Expression (',' Expression)? ')'
[59]	*IRIrefOrFunction*	::=	IRIref ArgList?
[60]	*RDFLiteral*	::=	String (LANGTAG | ('^^' IRIref))?
[61]	*NumericLiteral*	::=	NumericLiteralUnsigned | NumericLiteralPositive | NumericLiteralNegative
[62]	*NumericLiteralUnsigned*	::=	INTEGER | DECIMAL | DOUBLE
[63]	*NumericLiteralPositive*	::=	INTEGER_POSITIVE | DECIMAL_POSITIVE | DOUBLE_POSITIVE
[64]	*NumericLiteralNegative*	::=	INTEGER_NEGATIVE | DECIMAL_NEGATIVE | DOUBLE_NEGATIVE
[65]	*BooleanLiteral*	::=	'true' | 'false'

[66]	*String*	::=	STRING_LITERAL1 | STRING_LITERAL2 | STRING_LITERAL_LONG1 | STRING_LITERAL_LONG2
[67]	*IRIref*	::=	IRI_REF | PrefixedName
[68]	*PrefixedName*	::=	PNAME_LN | PNAME_NS
[69]	*BlankNode*	::=	BLANK_NODE_LABEL | ANON
[70]	*IRI_REF*	::=	'<' ([^<>"{}|^`\]-[#x00-#x20])* '>'
[71]	*PNAME_NS*	::=	PN_PREFIX? ':'
[72]	*PNAME_LN*	::=	PNAME_NS PN_LOCAL
[73]	*BLANK_NODE_LABEL*	::=	'_:' PN_LOCAL
[74]	*VAR1*	::=	'?' VARNAME
[75]	*VAR2*	::=	'$' VARNAME
[76]	*LANGTAG*	::=	'@' [a-zA-Z]+ ('-' [a-zA-Z0-9]+)*
[77]	*INTEGER*	::=	[0-9]+
[78]	*DECIMAL*	::=	[0-9]+ '.' [0-9]* | '.' [0-9]+
[79]	*DOUBLE*	::=	[0-9]+ '.' [0-9]* EXPONENT | '.' ([0-9])+ EXPONENT | ([0-9])+ EXPONENT
[80]	*INTEGER_POSITIVE*	::=	'+' INTEGER
[81]	*DECIMAL_POSITIVE*	::=	'+' DECIMAL
[82]	*DOUBLE_POSITIVE*	::=	'+' DOUBLE
[83]	*INTEGER_NEGATIVE*	::=	'-' INTEGER
[84]	*DECIMAL_NEGATIVE*	::=	'-' DECIMAL
[85]	*DOUBLE_NEGATIVE*	::=	'-' DOUBLE
[86]	*EXPONENT*	::=	[eE] [+-]? [0-9]+
[87]	*STRING_LITERAL1*	::=	"'" (([^#x27#x5C#xA#xD]) | ECHAR)* "'"
[88]	*STRING_LITERAL2*	::=	'"' (([^#x22#x5C#xA#xD]) | ECHAR)* '"'
[89]	*STRING_LITERAL_LONG1*	::=	"'''" (("'" | "''")? ([^'\] | ECHAR))* "'''"
[90]	*STRING_LITERAL_LONG2*	::=	'"""' (('"' | '""')? ([^"\] | ECHAR))* '"""'
[91]	*ECHAR*	::=	'\' [tbnrf\"']
[92]	*NIL*	::=	'(' WS* ')'
[93]	*WS*	::=	#x20 | #x9 | #xD | #xA
[94]	*ANON*	::=	'[' WS* ']'
[95]	*PN_CHARS_BASE*	::=	[A-Z] | [a-z] | [#x00C0-#x00D6] | [#x00D8-#x00F6] | [#x00F8-#x02FF] | [#x0370-#x037D] | [#x037F-#x1FFF] | [#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF] | [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD] | [#x10000-#xEFFFF]
[96]	*PN_CHARS_U*	::=	PN_CHARS_BASE | '_'
[97]	*VARNAME*	::=	(PN_CHARS_U | [0-9]) (PN_CHARS_U | [0-9] | #x00B7 | [#x0300-#x036F] | [#x203F-#x2040])*
[98]	*PN_CHARS*	::=	PN_CHARS_U | '-' | [0-9] | #x00B7 | [#x0300-#x036F] | [#x203F-#x2040]
[99]	*PN_PREFIX*	::=	PN_CHARS_BASE ((PN_CHARS|'.')* PN_CHARS)?
[100]	*PN_LOCAL*	::=	(PN_CHARS_U | [0-9]) ((PN_CHARS|'.')* PN_CHARS)?

References

[SPARQL-Q]
SPARQL Query Language for RDF, Eric Prud'hommeaux, Andy Seaborne (editors), W3C Recommendation.

[SPARQL-R]
SPARQL Query Results XML Format , D. Beckett (editor), W3C Recommendation

[RFC3986]
RFC 3986 *Uniform Resource Identifier (URI): Generic Syntax*, T. Berners-Lee, R.

Fielding, L. Masinter January 2005

[RFC3987]

RFC 3987, "Internationalized Resource Identifiers (IRIs)", M. Dürst , M. Suignard

[UNICODE]

The Unicode Standard, Version 4. ISBN 0-321-18578-1, as updated from time to time by the publication of new versions. The latest version of Unicode and additional information on versions of the standard and of the Unicode Character Database is available at http://www.unicode.org/unicode/standard/versions/.