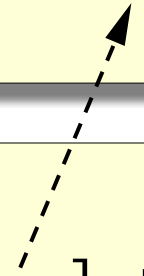
Quiz

Quiz

example.json

```
{
  "aaa" : {
            "bbb" : [11, 22]
          }
}
```



exampleRML.ttl

```
ex:tm a rml:TriplesMap ;

   rml:logicalSource [ rml:source [ … ] ;
                       rml:referenceFormulation rml:JSONPath ;
                       rml:iterator "$.aaa" ] ;

   rml:subjectMap [ … ] ;

   rml:predicateObjectMap [
            rml:predicateMap [ … ] ;

            rml:objectMap [ rml:reference "bbb" ;
                            rml:datatype ex:test ] ] .
```

Quiz

example.json

```
{
  "aaa" : {
            "bbb" : [11, 22]
          }
}
```

exampleRML.ttl

```
ex:tm a rml:TriplesMap ;

   rml:logicalSource [ rml:source [ … ] ;
                       rml:referenceFormulation rml:JSONPath ;
                       rml:iterator "$.aaa" ] ;

   rml:subjectMap [ … ] ;

   rml:predicateObjectMap [
            rml:predicateMap [ … ] ;

            rml:objectMap [ rml:reference "bbb" ;
                            rml:datatype ex:test ] ] .
```

my expectation:	one literal:
	`"[11, 22]"^^ex:test`

Quiz

example.json

```
{
  "aaa" : {
            "bbb" : [11, 22]
          }
}
```

exampleRML.ttl

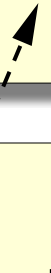
```
e [ … ] ;
enceFormulation rml:JSONPath ;
tor "$.aaa" ] ;

. ] ;

reference "bbb" ;
datatype ex:test ] ] .
```

System	Output for Test
RMLMapper (v 8.1.0)	two literals: "11"^^ex:test "22"^^ex:test
CARML (v 1.4.0-0.4.11)	two literals: "11"^^ex:test "22"^^ex:test
Morph-KGC (v 2.10.0)	two literals: "11"^^ex:test "22"^^ex:test
BURP (v 0.1.1)	one literal: "[11, 22]"^^ex:test
my expectation:	one literal: "[11, 22]"^^ex:test

Quiz 2

example.json

```
{
  "aaa" : {
            "bbb" : { "ccc" : 42,
                      "ddd" : [11,22] }
          }
}
```



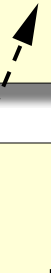
exampleRML.ttl

```
ex:tm a rml:TriplesMap ;

   rml:logicalSource [ rml:source [ … ] ;
                       rml:referenceFormulation rml:JSONPath ;
                       rml:iterator "$.aaa" ] ;

   rml:subjectMap [ … ] ;

   rml:predicateObjectMap [
            rml:predicateMap [ … ] ;

            rml:objectMap [ rml:reference "bbb" ;
                            rml:datatype ex:test ] ] .
```

Quiz 2

example.json

```
{
  "aaa" : {
            "bbb" : { "ccc" : 42,
                      "ddd" : [11,22] }
          }
}
```



exampleRML.ttl

```
ex:tm a rml:TriplesMap ;

   rml:logicalSource [ rml:source [ … ] ;
                       rml:referenceFormulation rml:JSONPath ;
                       rml:iterator "$.aaa" ] ;

   rml:subjectMap [ … ] ;

   rml:predicateObjectMap [
            rml:predicateMap [ … ] ;

            rml:objectMap [ rml:reference "bbb" ;
                            rml:datatype ex:test ] ] .
```

my expectation	one literal: `"""{"ccc": 42, "ddd": [11,22]}"""^^ex:test`

Quiz 2

example.json

```
{
  "aaa" : {
            "bbb" : { "ccc" : 42,
                      "ddd" : [11,22] }
          }
}
```

exampleRML.ttl

```
ex:tm a rml:TriplesMap ;
```

```
...ONPath ;
```

System	Output for Test 2
RMLMapper (v 8.1.0)	nothing
CARML (v 1.4.0-0.4.11)	one literal: `"""{ccc=42, ddd=[11, 22]}"""^^ex:test`
Morph-KGC (v 2.10.0)	one literal: `"""{ccc=42, ddd=[11, 22]}"""^^ex:test`
BURP (v 0.1.1)	one literal: `"""{ccc=42, ddd=[11, 22]}"""^^ex:test`
my expectation	one literal: `"""{"ccc": 42, "ddd": [11,22]}"""^^ex:test`

The HeFQUIN Approach to Extract and Process Composite Values with RML and SPARQL

Olaf Hartig

Department of Computer and Information Science (IDA)
Linköping University

LINKÖPING UNIVERSITY

The HeFQUIN Approach to Extract and Process Composite Values with RML and SPARQL





Olaf Hartig

Department of Computer and Information Science (IDA)
Linköping University

LINKÖPING UNIVERSITY

Unofficial Draft

TABLE OF CONTENTS

Abstract

Status of This Document

1. Introduction
1.1 Background and Motivation
1.2 Structure of this Document (TODO)
1.3 Document Conventions (TODO)
1.4 Conformance

2. Informal Description of the Approach
2.1 Representation of Composite Values as RDF Literals
2.2 SPARQL Functions on Composite Datatype Literals
2.3 FOLD Aggregate
2.4 UNFOLD Operator
2.5 Comparing Composite Datatype Literals
2.6 Ordering Composite Datatype Literals

3. The cdt:List Datatype
3.1 Value Space
3.2 Lexical Space
3.3 Lexical-To-Value Mapping
3.4 Relationship to RDF Collections

4. The cdt:Map Datatype
4.1 Value Space
4.2 Lexical Space
4.3 Lexical-To-Value Mapping
4.4 Relationship to JSON

5. Importing and Exporting of Composite Values
5.1 Motivation
5.2 Importing Requirements
5.3 Exporting Requirements

6. Extensions of Existing SPARQL Operators

SPARQL CDTs: Representing and Querying Lists and Maps as RDF Literals

Unofficial Draft 04 June 2024

link to spec

▾ **More details about this document**

Latest published version:
https://w3id.org/awslabs/neptune/SPARQL-CDTs/spec/latest.html

Latest editor's draft:
https://w3id.org/awslabs/neptune/SPARQL-CDTs/spec/editors_draft.html

History:
Commit history

Test suite:
https://github.com/awslabs/SPARQL-CDTs/tree/main/tests

Editors:
Olaf Hartig (Amazon)
Gregory Todd Williams (Amazon)

Feedback:
GitHub awslabs/SPARQL-CDTs (pull requests, new issue, open issues)

Copyright © 2024 the document editors/authors. Text is available under the Creative Commons Attribution 4.0 International Public License; additional terms may apply.

Abstract

This specification defines an approach to represent generic forms of composite values (lists and maps, in particular) as literals in RDF, and corresponding extensions of the SPARQL language. These extensions include an aggregation function to produce such composite values, functions to operate on such composite values in expressions, and a new operator to transform such composite values into their individual components.

Status of This Document

This document is a draft of a potential specification. It has no official standing of any kind and does not represent the support or consensus of any standards organization.

Lists and Maps as RDF Literals

link to spec

- cdt:List datatype
  - Value space: any finite sequence of RDF terms (and null)
  - Lexical space: based on Turtle, including short forms for literals
  - Example:

```
"[ <http://example.org>, 'hello'@en, 42, 43 ]"^^cdt:List
```

`xsd:integer` literal with value 42

- cdt:Map datatype
  - Value space: key-value pairs, with IRIs and literals as keys and RDF terms (and null) as values
  - Lexical space: also based on Turtle & designed to cover JSON
  - Example:

```
"{ <http://w3.org/> : 2, 'a' : {'b': [1,2]} }"^^cdt:Map
```

nesting possible

Quiz Revisited

example.json

```
{
  "aaa" : {
            "bbb" : [11, 22]
          }
}
```

exampleRML.ttl

```
ex:tm a rml:TriplesMap ;

   rml:logicalSource [ rml:source [ … ] ;
                       rml:referenceFormulation rml:JSONPath ;
                       rml:iterator "$.aaa" ] ;

   rml:subjectMap [ … ] ;

   rml:predicateObjectMap [
            rml:predicateMap [ … ] ;

            rml:objectMap [ rml:reference "bbb" ;
                            rml:datatype cdt:List ] ] .
```

Proposed result	one literal: "[11, 22]"^^cdt:List

Quiz Revisited

example.json

```
{
  "aaa" : {
            "bbb" : { "ccc" : 42,
                      "ddd" : [11,22] }
          }
}
```

exampleRML.ttl

```
ex:tm a rml:TriplesMap ;

   rml:logicalSource [ rml:source [ … ] ;
                       rml:referenceFormulation rml:JSONPath ;
                       rml:iterator "$.aaa" ] ;

   rml:subjectMap [ … ] ;

   rml:predicateObjectMap [
            rml:predicateMap [ … ] ;

            rml:objectMap [ rml:reference "bbb" ;
                            rml:datatype cdt:Map ] ] .
```

Proposed result	one literal:
	"""{"ccc": 42, "ddd": [11,22]}"""^^cdt:Map

SPARQL Extensions for CDT Literals

link to spec

- Functions to be used in expressions (FILTER, BIND, …)
  - get, size
  - concat, contains, head, reverse, subseq, tail,
  - containsKey, keys, merge, put, remove
  - constructors

- UNFOLD operator to decompose composite values

```
SELECT * WHERE { ?s :pred ?list
                 UNFOLD(?list AS ?elmt, ?pos) }
ORDER BY ?s ?pos
```

- FOLD aggregate to produce composite values

Demo

```
SELECT * WHERE {
  SERVICE <http://dbpedia.org/sparql>
  {
      dbp:Dubrovnik geo:lat ?lat ; geo:long ?long .
  }

  SERVICE <http://example.org/OpenMeteoWrapper>
          PARAMS( ?lat AS "latitude", ?long AS "longitude" )
  {
     ?x ex:hourlyTemperature ?list .
  }

  BIND ( cdt:get(?list, 1)          AS ?firstElmt )

  BIND ( cdt:size(?list)            AS ?listSize )
  BIND ( cdt:get(?list, ?listSize) AS ?lastElmt )

  BIND ( ?lastElmt - ?firstElmt AS ?tempDiff )
}
```

Demo

```
SELECT * WHERE {
  SERVICE <http://dbpedia.org/sparql>
  {
      dbp:Dubrovnik geo:lat ?lat ; geo:long ?long .
  }

  SERVICE <http://example.org/OpenMeteoWrapper>
          PARAMS( ?lat AS "latitude", ?long AS "longitude" )
  {
     ?x current ?c .
  }

  BIND ( cdt:keys(?c) AS ?keys )

  UNFOLD( ?keys AS ?key )
}
```

www.liu.se

LINKÖPING UNIVERSITY