# RDF Isolation API

James Leigh and David Wood, Zepheira

{jleigh, dwood}@zepheira.com

## *Introduction*

This document proposes a RESTful API for RDF stores that supports revision isolation. This proposed API combines basic CRUD operations over RDF services and queries and mandates RDF descriptions of services. The SPARQL 1.1 protocol includes the ability to modify an RDF store's state. With the ability to change state comes the challenge of managing store versions and the need to manage those versions (and their differences) over HTTP.

The SERVICE keyword is part of SPARQL 1.1 Federation Extensions[1] and is used here to link named queries and services. In this way the SPARQL language becomes the hypermedia linking services and graphs together.

An RDF store may have zero or more services. A service is an exposed branch of an RDF store. Just as a version control repository can have multiple parallel branches sharing much of the data, so too can conceptual RDF stores. Each service can have zero or more graphs, each graph can have zero or more triples.

Because each query, service, and graph are self describing they can exist in a peer-distributed network interlinked using the SERVICE and GRAPH keywords.

## *Background*

Other RESTful APIs for RDF have been proposed and implemented. Notable among these are the RESTful API to the Mulgara Semantic Store [2], the more recent Linked Data API proposed at the Second Linked Data Meetup London [3] and Pubby [4].

Mulgara's REST API allows basic CRUD operations via the issuing of HTTP verbs to URIs addressing RDF statements or graphs. Mulgara implements its REST operations by calling either its TQL query language features (that allow full CRUD operations) or its SPARQL endpoint (that has until recently been read-only). By contrast, the proposed Linked Data API and Pubby operate over SPARQL endpoints and provide read-only interfaces to RDF data. Pubby and the proposed Linked Data API address objects (in a Linked Data sense) by URI and map them to parameterized SPARQL queries.

The REST API proposed in this document is intended to provide a consistent, fully RESTful API to RDF data that supports CRUD operations at the query, graph and service levels and revision isolation. SPARQL 1.1 is used to define both queries and update graphs where necessary.

## *Requirements*

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocols it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements for its protocols is said to be "conditionally compliant."

All RDF results MUST be able to respond with application/rdf+xml if at all and MAY respond with text/turtle or other RDF type. All operations that respond with the application/sparql-query content type MUST return application/sparql-query if the request prefers it as the response type. All operation that POST or PUT the

application/sparql-query content type MUST accept application/sparql-query if anything.

Throughout this document the example URI paths such as /query, /branch and /service should be replaced with actual non-trivial URI paths. The example paths are meant to be just that and should not be taken literally. This document contains no “well-known locations;” any resource may exist on any HTTP accessible host.

All responses MAY respond with 401 Not Authorized and 403 Forbidden instead. All 2xx responses to GET, HEAD, PUT, and POST must include an Etag.

## *Named Query Resources*

## Create a Named Query

PUT of a SPARQL query SHOULD create a new named query. The "SERVICE" keywords are used to indicate which store branch the query will use during evaluation. In the absence of SERVICE keywords the default service should be used. The cache-control request header on PUT may be used in subsequent GET responses. Update queries MUST not be permitted as named queries.

Query parameters can be of the form <$name>, "$name", or "$name"^^<datatype>. URI bindings values are relative to the named query uri. If a binding has no value, it never matches. If a binding has multiple values, any and all values could match.

| Example Request | Example Response |
|---|---|
| PUT /query HTTP/1.1<br>Content-Type: application/sparql-query<br>Cache-Control: max-age=30<br><br>DESCRIBE ?book<br>FROM </graph><br>WHERE {<br> SERVICE </service> {<br>  ?book dc:creator "$author" .<br> }<br>} | HTTP/1.1 201 Created |

## View a Named Query

GET on a query SHOULD returns the query itself, along with the service(s) it uses; if agent prefers it.

| Example Request | Example Response |
|---|---|
| GET /query HTTP/1.1<br>Accept: application/sparql-query | HTTP/1.1 200 Ok<br>Content-Type: application/sparql-query<br><br>DESCRIBE ?book<br>FROM </graph><br>WHERE {<br> SERVICE </service> {<br>  ?book dc:creator "$author”<br> }<br>} |

## View the Results of a Named Query

GET of a SPARQL query with an accept type of boolean, tuples or RDF (other than application/sparql-query) SHOULD evaluate the query using a configured cache-control or responds with a previous evaluation. The query must be evaluated as if all combinations of query parameters were used in a BINDINGS clause[5]. The Age header must be used to indicate how long ago these results were evaluated. If the query cannot produce the expected output (indicated by the request Accept header) of 406 Not Acceptable is returned. This response and other responses MAY be the result of a redirect. The Etag used in the response must change every time the

results change as specified by HTTP.

| Example Request | Example Response |
|---|---|
| `GET /query?author=A.N.Other HTTP/1.1`<br>`Accept: text/turtle` | `HTTP/1.1 200 Ok`<br>`Content-Type: text/turtle`<br>`Cache-Control: max-age=30`<br>`Age: 0`<br><br>`<http://example.com/book3> dc:title "A new book" ;`<br>`  dc:creator "A.N.Other" .` |

## Evaluate Named Query

POST to a SPARQL query SHOULD evaluate the query serially using the bindings provided. The query must be evaluated as if all combinations of query parameters were used in a BINDINGS clause[5].

| Example Request | Example Response |
|---|---|
| `POST /query HTTP/1.1`<br>`Accept: text/turtle`<br>`Content-Type: application/x-www-form-urlencoded`<br><br>`author=A.N.Other` | `HTTP/1.1 200 OK`<br>`Content-Type: text/turtle`<br>`Cache-Control: no-store`<br><br>`<http://example.com/book3> dc:title "A new book" ;`<br>`  dc:creator "A.N.Other" .` |

## Remove Named Query

DELETE of a query MUST remove the query or respond with 401/403.

| Example Request | Example Response |
|---|---|
| `DELETE /query HTTP/1.1` | `HTTP/1.1 204 No Content` |

## *Store Service*

## Create a New Service

PUT of a SPARQL insert MUST create a new RDF store branch at the target URI that is initialized with the request body or respond with 401/403. If no SERVICE keyword is used the branch is a branch of the default service or an empty store. A service graph is automatically created when the service is created, this graph cannot be removed.

| Example Request | Example Response |
|---|---|
| `PUT /service HTTP/1.1`<br>`Content-Type: application/sparql-query`<br><br>`PREFIX dc:<http://purl.org/dc/elements/1.1/>`<br>`INSERT DATA {`<br>` <http://example.com/book3> dc:title "A new book";`<br>`  dc:creator "A.N.Other" .`<br>`}` | `HTTP/1.1 201 Created` |

## View All Data

GET of a service SHOULD returns the entire contents of the service. The commands in this response if replayed would restore the service to its current state.

| Example Request | Example Response |
|---|---|
| GET /service HTTP/1.1<br>Accept: application/sparql-query | HTTP/1.1 200 OK<br>Content-Type: application/sparql-query<br><br>PREFIX dc: <http://purl.org/dc/elements/1.1/><br>CREATE SILENT GRAPH <><br>WITH <><br>INSERT DATA {<br> <http://example.com/book3> dc:title "A new book";<br> dc:creator "A.N.Other" .<br>} |

## View Description

GET of a service for RDF (of a format that does not support named graphs such as rdf+xml and turtle) MUST return the contents of the named graph of the service name. This named graph SHOULD be used to describe the service, related services, available named queries, and relevant named graphs.

Any queries referenced in this service graph MUST implement View the Results of a Named Query. An RDF vocabulary to describe services must be developed for this function to be completely defined.

Implementation of the View Description function is REQUIRED.

| Example Request | Example Response |
|---|---|
| GET /service HTTP/1.1<br>Accept: application/rdf+xml | HTTP/1.1 200 OK<br>Content-Type: application/rdf+xml<br><br><rdf:RDF<br> xmlns:dc="http://purl.org/dc/elements/1.1/"<br> xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"><br><rdf:Description<br> rdf:about="http://example.com/book3"><br> <dc:title>A new book</dc:title><br> <dc:creator>A.N.Other</dc:creator><br></rdf:Description><br></rdf:RDF> |

## Evaluate Query

POST to a service MUST evaluate the query serially. This is provided (in contrast to named queries) to ensure that queries are evaluated in the correct sequence among other update operations.

Implementation of the Evaluate Query function is REQUIRED.

| Example Request | Example Response |
|---|---|
| POST /service HTTP/1.1<br>Accept: text/turtle<br>Content-Type: application/sparql-query<br><br>DESCRIBE ?book<br>WHERE {<br> ?book dc:title "A new book" ;<br> dc:creator "A.N.Other" .<br>} | HTTP/1.1 200 OK<br>Content-Type: text/turtle<br><br><http://example.com/book3><br> dc:title "A new book" ;<br> dc:creator "A.N.Other" . |

## Modify State

POST of an update to a service MUST evaluate the operation serially or return 403 Forbidden. Inserted triples default to the service graph unless otherwise indicated using the WITH keyword. Inserted triples default to being stored in the current service unless otherwise indicated using the WITH SERVICE keywords.

| Example Request | Example Response |
| --- | --- |
| POST /service HTTP/1.1<br>Accept: text/turtle<br>Content-Type: application/sparql-query<br><br>PREFIX dc: <http://purl.org/dc/elements/1.1/><br>INSERT DATA {<br>  <http://example.com/book3> dc:title "A book" ;<br>  dc:creator "A.Other" .<br>} | HTTP/1.1 204 No Content |

## Remove Service

DELETE to a service MUST remove the branch from the RDF store or respond with 401 Not Authorized or 403 Forbidden.

| Example Request | Example Response |
| --- | --- |
| DELETE /service HTTP/1.1 | HTTP/1.1 204 No Content |

## *Virtual Store Service*

## Branch a Service

PUT (with SERVICE) MAY branch an existing store revision to a new store revision (a copy). Although a copy is made available before the 201 response is received, there is no guarantee when the state will be copied. The copied state maybe a copy of the past, present, or future state of the service.

| Example Request | Example Response |
| --- | --- |
| PUT /branch HTTP/1.1<br>Content-Type: application/sparql-query<br><br>INSERT { ?s ?p ?o }<br>WHERE {<br>  SERVICE </service> {<br>  ?s ?p ?o<br>  }<br>} | HTTP/1.1 201 Created |

## *Include a Service*

POST (with SERVICE) MAY incorporate the content of another service into an already existing service.

| Example Request | Example Response |
| --- | --- |
| POST /branch HTTP/1.1<br>Content-Type: application/sparql-query<br><br>INSERT { ?s ?p ?o }<br>WHERE {<br>  SERVICE </service> {<br>  ?s ?p ?o<br>  }<br>} | HTTP/1.1 204 Not Content |

## Branch a Dataset

A service may constrain the triples that are included from the other service. The new service will only expose matching triples in the target service.

| Example Request | Example Response |
| --- | --- |
| PUT /branch HTTP/1.1<br>Content-Type: application/sparql-query<br><br>INSERT { ?s ?p ?o }<br>WHERE {<br>  SERVICE </service> {<br>  ?s ?p ?o<br>  FILTER regex(str(?s), “^http://example.com/”)<br>  }<br>} | HTTP/1.1 201 Created |

## View Changes

GET of a service SHOULD only return the delta changes since the branch was created and what services were copied. The service response contains a sequence of commands that recreate the current state of the service.

| Example Request | Example Response |
| --- | --- |
| GET /branch HTTP/1.1<br>Accept: application/sparql-query | HTTP/1.1 200 OK<br>Content-Type: application/sparql-query<br><br>PREFIX dc: <http://purl.org/dc/elements/1.1/><br>CREATE SLIENT GRAPH <><br>INSERT { ?s ?p ?o }<br>WHERE {<br>  SERVICE </service> {<br>  ?s ?p ?o<br>  }<br>}<br>WITH <><br>INSERT DATA {<br> <http://example.com/book3> dc:title "A new book";<br>  dc:creator "A.N.Other" .<br>} |

## Proxy changes

POST of an update to a service MAY proxy changes to a backing service if the changes would be viewable in this service.

| Example Request | Example Response |
| --- | --- |
| POST /service HTTP/1.1<br>Accept: text/turtle<br>Content-Type: application/sparql-query<br><br>PREFIX dc: <http://purl.org/dc/elements/1.1/><br>WITH SERVICE </service><br>INSERT DATA {<br>  <http://example.com/book3> dc:title "A book" ;<br>  dc:creator "A.Other" .<br>} | HTTP/1.1 204 No Content |

## Verify Revision

POST with a content-location of a revision branch MAY return 202 Accepted if the observed content revision state is consistent and the agent should resend the request to complete the merge.

| Example Request | Example Response |
| --- | --- |
| POST /service HTTP/1.1<br>Content-Location: /branch<br>Content-Type: application/sparql-query<br><br>PREFIX dc: <http://purl.org/dc/elements/1.1/><br>INSERT { ?s ?p ?o }<br>WHERE {<br>  SERVICE </service> {<br>  ?s ?p ?o<br>  }<br>}<br>CREATE SLIENT GRAPH <><br>WITH <><br>INSERT DATA {<br> <http://example.com/book3> dc:title "A new book";<br>  dc:creator "A.N.Other" .<br>} | HTTP/1.1 202 Accepted |

## Apply Revision

POST with a content-location of a revision branch MAY not succeed if the observed content revision state is inconsistent with the target revision state otherwise the service should be merged and 204 response is given.

| Example Request | Example Response |
| --- | --- |
| POST /service HTTP/1.1<br>Content-Location: /branch<br>Content-Type: application/sparql-query<br><br>PREFIX dc: <http://purl.org/dc/elements/1.1/><br>INSERT { ?s ?p ?o }<br>WHERE {<br>  SERVICE </service> {<br>  ?s ?p ?o<br>  }<br>}<br>CREATE SLIENT GRAPH <><br>WITH <><br>INSERT DATA {<br> <http://example.com/book3> dc:title "A new book";<br>  dc:creator "A.N.Other" .<br>} | HTTP/1.1 204 No Content |

## *Graph Services*

## Create a Graph Service

A service can expose a named graph of the same name in a different service.

| Example Request | Example Response |
| --- | --- |
| PUT /graph HTTP/1.1<br>Content-Type: application/sparql-query<br><br>PREFIX dc: <http://purl.org/dc/elements/1.1/><br>INSERT { ?s ?p ?o }<br>WHERE {<br>  SERVICE </service> {<br>  SELECT ?s ?p ?o {<br>  GRAPH <> { ?s ?p ?o }<br>  }<br>  }<br>} | HTTP/1.1 201 Created |

## View Graph

GET of a service MUST returns the contents of the service graph, which may be stored in another service.

| Example Request | Example Response |
|---|---|
| `GET /graph HTTP/1.1`<br>`Accept: application/rdf+xml` | `HTTP/1.1 200 OK`<br>`Content-Type: application/rdf+xml`<br><br>`<rdf:RDF`<br>`  xmlns:dc="http://purl.org/dc/elements/1.1/"`<br>`  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">`<br>`<rdf:Description`<br>`  rdf:about="http://example.com/book3">`<br>`  <dc:title>A new book</dc:title>`<br>`  <dc:creator>A.N.Other</dc:creator>`<br>`</rdf:Description>`<br>`</rdf:RDF>` |

## Update Graph Data

POST to a service MAY be applied using the default graph (same name as target), but against another service (provided the other service is visible from the target service).

| Example Request | Example Response |
|---|---|
| `POST /graph HTTP/1.1`<br>`Content-Type: application/sparql-query`<br><br>`WITH SERVICE </service>`<br>`DELETE {`<br>`  ?book dc:title "A new book" ;`<br>`  dc:creator "A.N.Other" .`<br>`}`<br>`WITH SERVICE </service>`<br>`INSERT DATA {`<br>`  <http://example.com/book3> dc:title "A new book";`<br>`  dc:creator "A.N.Other" .`<br>`}` | `HTTP/1.1 204 No Content` |

## Remove Graph Service

DELETE to a service MUST not modify any other services and MUST only remove the service and any state stored exclusively within it. In the case of a remote service graph, the remote graph is not removed.

| Example Request | Example Response |
|---|---|
| `DELETE /graph HTTP/1.1` | `HTTP/1.1 204 No Content` |

## *References*


[1] http://www.w3.org/TR/sparql11-federated-query/

[2] Paul Gearon, Mulgara REST API, http://www.mulgara.org/trac/wiki/RESTInterface

[3] Dave Reynolds, Jeni Tennison and Leigh Dodds, Linked Data API: http://code.google.com/p/linked-data-api/wiki/Specification

[4] Richard Cyganiak and Chris Bizer, Pubby: http://www4.wiwiss.fu-berlin.de/pubby/

[5] http://www.w3.org/TR/sparql11-query/#rBindingsClause