Voldemort thin client REST API

REST API spec

Introduction

URI Specification

Objects stored in VOLDEMORT are identified by the following URI space:

```
http://cluster.voldemort.com/<store name>/<Key>{,<Key>}
```

Where

- <voldemort-cluster-name> is the specific name of the Voldemort cluster (Eg: Vaas / Money01 / Locale / ...)
- <store name> is the name of the Voldemort store hosted on that particular cluster
- <Key> is the identifier by which resources (or values) within a Voldemort store are referenced i.e. a Key.

To minimize round-trips from the client to the Voldemort coordinator, and to multiplex multiple resource requests over a single socket connection, the API permits a comma delimited list of Keys.

Semantically, the following request

```
GET /pymk/132123,934832,345345
```

is equivalent to issuing the following sequence of requests:

```
GET /pymk/132123
GET /pymk/934832
GET /pymk/345345
```

Schema Specification

TBD

Schema Evolution

TODO: Complete this section

Object Representation

Voldemort keys and values can be stored in a variety of serialization formats: <java-serialization, string, identity, json, protobuf, thrift, avro-generic, avro-specific, avro-reflective, avro-generic-versioned>
In HTTP requests and responses, Voldemort supports JSON serialization only at first. The keys and values are both UNICODE encoded.

Common Request Headers

Header Name	Description	Required	PUT	GET / HEAD	DELETE
Accept	Specifies the return types the client is prepared to accept. Allowed values & Default: application/json	No	ignored	optional	ignored
Accept-Encoding	Specifies the content-encodings that the client will accept Allowed Values: `gzip` Default: None	No	ignored	optional	ignored

Authorization	TBD: For use in future multi-tennant releases to authenticate callers	<future>	<future>	<future>	<future>
Cache-Control	Specifies restrictions cached representations that are acceptable to the client Default: None	No	ignored	optional	ignored
Content-Type	The content type of the resource. Example: `application/json` Default: None Condition: Required for PUT requests	Conditional	required	ignored	ignored
X-VOLD-Vector-Clock	The vector clock for the corresponding value to be used in a PUT request. Default: None	No	optional	ignored	ignored
X-VOLD-Timestamp	The timestamp associated with the vector clock for the corresponding value to be used in a PUT request. Default: None	No	optional	ignored	ignored
X-VOLD-Request-Timeout	The timeout value (in milliseconds) to be used for the request. Default: None	Yes	required	required	required
X-VOLD-Inconsistency-Resolver	The type of inconsistency resolver to be used. Allowed values: custom Default: None	No	ignored	optional	ignored

Note1: If X-VOLD-Inconsistency-Resolver header is not specified, then the response will be returned after the default timestamp based resolution is applied to the raw results (which may contain multi-versioned values for the same key).
In other words, in the absence of this header, the thin client will always get only one value in the GET response (along with its corresponding Vector clock in ETag).

Note2: The Etag value received in a GET response can be used in the X-VOLD-Vector-Clock header in the subsequent PUT request.

Common Response Headers

Header Name	Description	Required	PUT	GET / HEAD	DELETE
Cache-Control	Specifies cached restrictions imposed by the server on the returned resource Default: None	No	N/A	optional	N/A
Content-Length	Length of the message (without headers) according to RFC 2616. Default: None Condition: Required for PUT requests	Conditional	N/A	required	N/A
Content-Location	URI for which key this part belongs to in a multi-part (during a getAll) response.	Conditional	N/A	multi-only	N/A
Content-Type	The content type of the resource. Allowed values & Default:: `application/json` Condition: Required for PUT requests	Conditional	N/A	required	N/A
Date	The date and time at which the response was generated. Note: this is **not** the last modification time of the resource	Yes	required	required	required
ETag	The Vector clock value for the returned value. This value is unique across all versioned values for a specific key.	Yes	required	required	N/A
Last-Modified	The last modification timestamp of the specified resource in HTTP-date format	Yes	required	required	N/A

Response headers listed as "N/A" are not returned by VOLDEMORT for the specified operation.

Response headers listed as "multi-only" are only returned in the individual MIME section for multi-part responses.

Operations Types:

DELETE

Deletes the key specified by the request URI.

Sample Request

```
DELETE /<store_name>/<key> HTTP/1.1
  Host: cluster.voldemort.com
  Content-Length: 0
  X-VOLD-Request-Timeout: NNNN
```

Sample Response

For a successful delete, the server will return:

```
HTTP/1.1 204 No Content
  Date: Wed, 09 Mar 2011 20:35:54 GMT
  Content-Length: 0
```

If the specified resource does note exist

```
HTTP/1.1 404 Not Found
  Date: Wed, 09 Mar 2011 20:35:54 GMT
  Content-Length: 0
```

GET

To fetch a value, simply perform an HTTP GET for the key referencing that value.

Sample Request

```
GET /<store-name>/<key> HTTP/1.1
  Host: cluster.voldemort.com
  X-VOLD-Request-Timeout: NNNN
  [Accept: application/json]                                                    # Only one
serialization type for now (also the default)
```

Sample Response

The return HTTP payload would be an Avro encoded blob:

```
HTTP/1.1 200 OK
   Date: Wed, 26 Jan 2011 19:54:11 GMT
   Last-Modified: Tue, 25 Jan 2011 13:05:24 GMT
   ETag: <Unicode encoded Vector clock>
   Content-Type: application/json
   Content-Transfer-Encoding: binary
   Content-Length: NNNN

   <NNNN bytes of binary value data>
```

HEAD

HEAD is used to retrieve metadata about a resource (value) without returning the resource itself.

Sample Request

```
HEAD /<store-name>/<key> HTTP/1.1
  Host: cluster.voldemort.com
  X-VOLD-Request-Timeout: NNNN
  [Accept: application/json]
```

Sample Response

```
HTTP/1.1 200 OK
   Date: Wed, 26 Jan 2011 19:54:11 GMT
   Last-Modified: Tue, 25 Jan 2011 13:05:24 GMT
   ETag: <Unicode encoded Vector clock>
   Content-Type: application/json
   Content-Length: NNNN
```

Note: The `Content-Length` returned is the size of the entity-body that would have been sent had the request been a GET (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.13).

PUT

Without a Vector clock in the Request

The PUT inserts a new resource (value) for a given key or replaces an existing resource (value) given the full resource ID. Since PUT requires a completely specified resource, the response to a PUT request does not include an entity-body containing the resource.

In this type, since the X-VOLD-Vector-Clock is not specified in the request, it is the responsibility of the receiver of this HTTP request to fetch the existing vector clock for this key (if any) and then do the update based on that Vector clock. If the key does not exist, then a new Vector clock is created by the receiver of this HTTP request.

Sample Request

```
PUT /<store-name>/<Unicode encoded key> HTTP/1.1
  Host: cluster.voldemort.com
  Content-Type: application/json
  Content-Length: 4
  X-VOLD-Request-Timeout: NNNN
  [Accept: application/json]

  {}
```

Sample Response

```
HTTP/1.1 201 Created
   Date: Tue, 10 Mar 2011 17:44:10 GMT
   Last-Modified: Tue, 10 Mar 2011 17:44:10 GMT
   ETag: <Unicode encoded Vector Clock>
   Content-Length: 0
```

With a Vector clock in the Request

This is similar to the previous PUT request but here the application is supplying a vector clock. The receiver of this HTTP request does not need to perform the extra step of fetching the existing vector clock for this key (if any).

Sample Request

```
PUT /<store-name>/<Unicode encoded key> HTTP/1.1
  Host: cluster.voldemort.com
  Content-Type: application/json
  Content-Length: 4
  X-VOLD-Request-Timeout: NNNN
  X-VOLD-Vector-Clock: <Unicode encoded Vector clock>
  X-VOLD-Timestamp: <Timestamp for this value in HTTP Date format>
  [Accept: application/json]

  {}
```

Sample Response

```
HTTP/1.1 201 Created
   Date: Tue, 10 Mar 2011 17:44:10 GMT
   Last-Modified: Tue, 10 Mar 2011 17:44:10 GMT
   ETag: <Unicode encoded Vector Clock>
   Content-Length: 0
```

Response Codes Returned

Code	Summary	Description
200	OK	An existing resource was successfully replaced with the content specified in the entity-body
201	Created	A new resource was successfully created
400	Bad Request	The request was malformed
500	Internal Server Error	<expletive> happens

GET multiple specific resources

Multiple resources may be specified in a single GET operation by replacing a single key with a parenthesized, comma delimited list of keys.

Sample Request

```
GET /<store-name>/(key1,key2,key3,key4) HTTP/1.1
   Host: cluster.voldemort.com
   X-VOLD-Request-Timeout: NNNN
   [Accept: application/json]
```

Sample Response

```
HTTP/1.1 200 OK
  Date: Thu, 10 Mar 2011 23:14:12 GMT
  Content-Length: 822
  Content-Type: multipart/binary; boundary=1296160692

  --1296160692
  Content-Type: application/json
  Content-Location: /<store-name>/key1
  Content-Transfer-Encoding: binary
  Content-Length: n1
  Last-Modified: Tue, 10 Mar 2011 17:44:10 GMT
  ETag: <Unicode encoded Vector Clock>
  <n1 bytes of binary data>

  --1296160692
  Content-Type: application/json
  Content-Location: /<store-name>/key2
  Content-Transfer-Encoding: binary
  Content-Length: n2
  Last-Modified: Mon, 9 Mar 2011 14:53:03 GMT
  ETag: <Unicode encoded Vector Clock>
  <n2 bytes of binary data>

  --1296160692
  Content-Type: application/json
  Content-Location: /<store-name>/key3
  Content-Transfer-Encoding: binary
  Content-Length: n3
  Last-Modified: Mon, 9 Mar 2011 04:19:21 GMT
  ETag: <Unicode encoded Vector Clock>
  <n3 bytes of binary data>

  --1296160692
  Content-Type: application/json
  Content-Location: /<store-name>/key4
  Content-Transfer-Encoding: binary
  Content-Length: n4
  Last-Modified: Sun, 8 Mar 2011 18:34:73 GMT
  ETag: <Unicode encoded Vector Clock>
  <n4 bytes of binary data>
  --1296160692--
```

GET all the versions of a particular resource

In case the client is using a custom consistency resolver, the client can request to receive all the versions of a resource (value) for a particular key and then resolve the inconsistency using the custom resolver code provided by the application. The request message is similar to a regular GET request in addition to an additional header:

Sample Request

```
GET /<store-name>/key HTTP/1.1
   Host: cluster.voldemort.com
   [Accept: application/json]
   X-VOLD-Request-Timeout: NNNN
   X-VOLD-Inconsistency-Resolver: custom
```

Sample Response

Assume that there was a conflict for this particular key: 'key'. The response is then multi-part with each part representing a particular version for that key. Following is a sample response with two versions.

```
HTTP/1.1 200 OK
  Date: Thu, 10 Mar 2011 23:14:12 GMT
  Content-Length: 822
  Content-Type: multipart/binary; boundary=1296160692

  --1296160692
  Content-Type: application/json
  Content-Location: /<store-name>/key
  Content-Transfer-Encoding: binary
  Content-Length: n1
  Last-Modified: Tue, 10 Mar 2011 17:44:10 GMT
  ETag: <Unicode encoded Vector Clock 1>
  <n1 bytes of binary data>

  --1296160692
  Content-Type: application/json
  Content-Location: /<store-name>/key
  Content-Transfer-Encoding: binary
  Content-Length: n2
  Last-Modified: Mon, 9 Mar 2011 14:53:03 GMT
  ETag: <Unicode encoded Vector Clock 2>
  <n2 bytes of binary data>
  --1296160692--
```

MULTI-GET with all the versions of a particular resource

Similar to GET with multiple versions, but for a list of keys:

Sample Request

```
GET /<store-name>/(key1,key2) HTTP/1.1
   Host: cluster.voldemort.com
   X-VOLD-Request-Timeout: NNNN
   [Accept: application/json]
   X-VOLD-Inconsistency-Resolver: custom
```

Sample Response

Assume that there was a conflict for this particular key: 'key'. The response is then a nested multi-part HTTP message. The first level represents the value for a particular key. The second level represents the particular version of a value for that particular key.

```
HTTP/1.1 200 OK
  Date: Thu, 10 Mar 2011 23:14:12 GMT
  Content-Length: 822
  Content-Type: multipart/binary; boundary=key-separator

  --key-separator
  Content-Type: multipart/binary; boundary=value-separator

  --value-separator
  Content-Type: application/json
  Content-Location: /<store-name>/key1
  Content-Transfer-Encoding: binary
  Content-Length: n1
  Last-Modified: Tue, 10 Mar 2011 17:44:10 GMT
  ETag: <Unicode encoded Vector Clock 1>

  <n1 bytes of binary data>

  --value-separator
  Content-Type: application/json
  Content-Location: /<store-name>/key1
  Content-Transfer-Encoding: binary
  Content-Length: n2
  Last-Modified: Mon, 9 Mar 2011 14:53:03 GMT
  ETag: <Unicode encoded Vector Clock 2>

  <n2 bytes of binary data>

  --value-separator--

  --key-separator
  Content-Type: multipart/binary; boundary=value-separator

  --value-separator
  Content-Type: application/json
  Content-Location: /<store-name>/key2
  Content-Transfer-Encoding: binary
  Content-Length: n3
  Last-Modified: Tue, 10 Mar 2011 17:44:10 GMT
  ETag: <Unicode encoded Vector Clock 3>

  <n3 bytes of binary data>

  --value-separator
  Content-Type: application/json
  Content-Location: /<store-name>/key2
  Content-Transfer-Encoding: binary
  Content-Length: n4
  Last-Modified: Mon, 9 Mar 2011 14:53:03 GMT
  ETag: <Unicode encoded Vector Clock 4>

  <n4 bytes of binary data>

  --value-separator--

  --key-separator--
```