ArangoDB's Reference Manual for API Implementors

The ArangoDB Team

Version

Contents

Chapter 1

REST Interface for Documents

1.1 Documents, Identifiers, Handles

This is an introduction to ArangoDB's REST interface for documents.

Document: Documents in ArangoDB are JSON objects. These objects can be nested (to any depth) and may contains lists. Each document is uniquely identified by its document handle. For example:

```
{
  "_id" : "1234567/2345678",
  "_rev" : "3456789",
  "firstName" : "Hugo",
  "lastName" : "Schlonz",
  "address" : {
    "street" : "Strasse 1",
    "city" : "Hier"
  },
  "hobbies" : [
    "swimming",
    "biking",
    "programming"
  ]
}
```

All documents contain special attributes: the document handle in `_id`, the document's unique key in `_key` and and the etag aka document revision in `_rev`. The value of the `_key` attribute can be specified by the user when creating a document. `_id` and `_key` values are immutable once the document has been created. The `_rev` value is maintained by ArangoDB autonomously.

Document Handle: A document handle uniquely identifies a document in the database. It is a string and consists of a collection name and a document key separated by `/`.

Document Key: A document key uniquely identifies a document in a given collection. It can and should be used by clients when specific documents are searched. Document keys are stored in the `_key` attribute of documents. The key values are automatically indexed by ArangoDB so looking up a document by its key is regularly a fast operation. The `_key` value of a document is immutable once the document has been created.

By default, ArangoDB will auto-generate a document key if no `_key` attribute is specified, and use the user-specified `_key` otherwise.

This behavior can be changed on a per-collection level by creating collections with the `keyOptions` attribute.

Using `keyOptions` it is possible to disallow user-specified keys completely, or to force a specific regime for auto-generating the `_key` values.

Document Revision: As ArangoDB supports MVCC, documents can exist in more than one revision. The document revision is the MVCC token used to identify a particular revision of a document. It is a string value currently containing an integer number and is unique within the list of document revisions for a single document. Document revisions can be used to conditionally update, replace or delete documents in the database. In order to find a particular revision of a document, you need the document handle and the document revision.

ArangoDB currently uses 64bit unsigned integer values to maintain document revisions internally. When returning document revisions to clients, ArangoDB will put them into a string to ensure the revision id is not clipped by clients that do not support big integers. Clients should treat the revision id returned by ArangoDB as an opaque string when they store or use it locally. This will allow ArangoDB to change the format of revision ids later if this should be required. Clients can use revisions ids to perform simple equality/non-equality comparisons (e.g. to check whether a document has changed or not), but they should not use revision ids to perform greater/less than comparisions with them to check if a document revision is older than one another, even if this might work for some cases.

Note: revision ids have been returned as integers up to including ArangoDB 1.1

Document Etag: The document revision enclosed in double quotes. The revision is returned by several HTTP API methods in the `Etag` HTTP header. The basic operations (create, read, update, delete) for documents are mapped to the standard HTTP methods (`POST`, `GET`, `PUT`, `DELETE`). There is also a partial update method, which is mapped to the HTTP `PATCH` method.

An identifier for the document revision is returned in the `ETag` header field. If you modify a document, you can use the `If-Match` field to detect conflicts. The revision of a document can be checking using the HTTP method `HEAD`.

1.2 Address and ETag of a Document

All documents in ArangoDB have a document handle. This handle uniquely defines a document and is managed by ArangoDB. All documents are found under the URI:

```
http://server:port/_api/document/document-handle
```

For example: Assume that the document handle, which is stored in the `_id` attribute of the document, is `demo/362549736`, then the URL of that document is:

```
http://localhost:8529/_api/document/demo/362549736
```

Each document also has a document revision or etag with is returned in the "ETag" header field when requesting a document.

If you obtain a document using `GET` and you want to check if a newer revision is available, then you can use the `If-None-Match` header. If the document is unchanged, a `HTTP 412` is returned.

If you want to update or delete a document, then you can use the `If-Match` header. If the document has changed, then the operation is aborted and a `HTTP 412` is returned.

1.3 Working with Documents using REST

GET /_api/document/`document-handle`,reads a document

document-handle (string,required) The Handle of the Document.

If-None-Match (string,optional) If the "If-None-Match" header is given, then it must contain exactly one etag. The document is returned, if it has a different revision than the given etag. Otherwise a `HTTP 304` is returned.

If-Match (string,optional) If the "If-Match" header is given, then it must contain exactly one etag. The document is returned, if it has the same revision ad the given etag. Otherwise a `HTTP 412` is returned. As an alternative you can supply the etag in an attribute `rev` in the URL.

Returns the document identified by `document-handle`. The returned document contains two special attributes: `_id` containing the document handle and `_rev` containing the revision.

is returned if the document was found

is returned if the document or collection was not found

is returned if the "If-None-Match" header is given and the document has same version

is returned if a "If-Match" header or `rev` is given and the found document has a different version

Use a document handle:

```
unix> curl --dump - http://localhost:8529/_api/
   document/products/30032295

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: "30032295"

{
  "hello" : "world",
  "_id" : "products/30032295",
  "_rev" : "30032295",
  "_key" : "30032295"
}
```

Use a document handle and an etag:

```
unix> curl --header 'If-None-Match:"31343015"' --dump
   - http://localhost:8529/_api/document/products
   /31343015

HTTP/1.1 304 Not Modified
```

```
content-type: text/plain; charset=utf-8
etag: "31343015"
```

Unknown document handle:

```
unix> curl --dump - http://localhost:8529/_api/
   document/products/unknownhandle

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8

{
  "error" : true,
  "errorMessage" : "document /_api/document/products/
     unknownhandle not found",
  "code" : 404,
  "errorNum" : 1202
}
```

POST /_api/document,creates a document

document (json,required) A JSON representation of document.

collection (string,required) The collection name.

createCollection (boolean,optional) If this parameter has a value of `true` or `yes`, then the collection is created if it does not yet exist. Other values will be ignored so the collection must be present for the operation to succeed.

waitForSync (boolean,optional) Wait until document has been sync to disk.

Creates a new document in the collection named `collection`. A JSON representation of the document must be passed as the body of the POST request.

If the document was created successfully, then the "Location" header contains the path to the newly created document. The "ETag" header field contains the revision of the document.

The body of the response contains a JSON object with the following attributes:

- `_id` contains the document handle of the newly created document
- `_key` contains the document key
- `_rev` contains the document revision

If the collection parameter `waitForSync` is `false`, then the call returns as soon as the document has been accepted. It will not wait, until the documents has been sync to disk.

Optionally, the URL parameter `waitForSync` can be used to force synchronisation of the document creation operation to disk even in case that the `waitForSync` flag had been disabled for the entire collection. Thus, the `waitForSync` URL parameter can be used to force synchronisation of just this specific operations. To use this, set the `waitForSync` parameter to `true`. If the `waitForSync` parameter is not specified or set to `false`, then the collection's default `waitForSync` behavior is applied. The `waitForSync` URL parameter cannot be used to disable synchronisation for collections that have a default `waitForSync` value of `true`.

is returned if the document was created sucessfully and `waitForSync` was `true`.

is returned if the document was created sucessfully and `waitForSync` was `false`.

is returned if the body does not contain a valid JSON representation of a document. The response body contains an error document in this case.

is returned if the collection specified by `collection` is unknown. The response body contains an error document in this case.

Create a document given a collection named `products`. Note that the revision identifier might or might not by equal to the auto-generated key.

```
unix> curl -X POST --data @- --dump - http://localhost
   :8529/_api/document?collection=products
{ "Hello": "World" }

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
etag: "24986023"
location: /_api/document/products/24986023

{
  "error" : false,
  "_id" : "products/24986023",
  "_rev" : "24986023",
  "_key" : "24986023"
}
```

Create a document in a collection named `products` with a collection-level `waitForSync` value of `false`.

```
unix> curl -X POST --data @- --dump - http://localhost
   :8529/_api/document?collection=products
{ "Hello": "World" }

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
etag: "26427815"
location: /_api/document/products/26427815

{
  "error" : false,
  "_id" : "products/26427815",
  "_rev" : "26427815",
  "_key" : "26427815"
}
```

Create a document in a collection with a collection-level `waitForSync` value of `false`, but using the `waitForSync` URL parameter.

```
unix> curl -X POST --data @- --dump - http://localhost
   :8529/_api/document?collection=products&waitForSync
   =true
{ "Hello": "World" }
```

```
HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
etag: "27541927"
location: /_api/document/products/27541927

{
  "error" : false,
  "_id" : "products/27541927",
  "_rev" : "27541927",
  "_key" : "27541927"
}
```

Create a document in a new, named collection

```
unix> curl -X POST --data @- --dump - http://localhost
   :8529/_api/document?collection=products&
   createCollection=true
{ "Hello": "World" }

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
etag: "28787111"
location: /_api/document/products/28787111

{
  "error" : false,
  "_id" : "products/28787111",
  "_rev" : "28787111",
  "_key" : "28787111"
}
```

Unknown collection name:

```
unix> curl -X POST --data @- --dump - http://localhost
   :8529/_api/document?collection=products
{ "Hello": "World" }

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8

{
  "error" : true,
  "errorMessage" : "collection /_api/collection/
     products not found",
  "code" : 404,
```

```
  "errorNum" : 1203
}
```

Illegal document:

```
unix> curl -X POST --data @- --dump - http://localhost
   :8529/_api/document?collection=products
{ 1: "World" }

HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8

{
  "error" : true,
  "errorMessage" : "expecting attribute name",
  "code" : 400,
  "errorNum" : 600
}
```

PUT /_api/document/`document-handle`,replaces a document

document-handle (string,required) The Handle of the Document.

waitForSync (boolean,optional) Wait until document has been sync to disk.

rev (string,optional) You can conditionally replace a document based on a target revision id by using the `rev` URL parameter.

policy (string,optional) To control the update behavior in case there is a revision mismatch, you can use the `policy` parameter. This is the same as when replacing documents (see replacing documents for more details).

If-Match (string,optional) You can conditionally replace a document based on a target revision id by using the `if-match` HTTP header.

Completely updates (i.e. replaces) the document identified by `document-handle`. If the document exists and can be updated, then a `HTTP 201` is returned and the "ETag" header field contains the new revision of the document.

If the new document passed in the body of the request contains the `document-handle` in the attribute `_id` and the revision in `_rev`, these attributes will be ignored. Only the URI and the "ETag" header are relevant in order to avoid confusion when using proxies.

Optionally, the URL parameter `waitForSync` can be used to force synchronisation of the document replacement operation to disk even in case that the `waitForSync` flag had been disabled for the entire collection. Thus, the `waitForSync` URL parameter can be used to force synchronisation of just specific operations. To use this, set the `waitForSync` parameter to `true`. If the `waitForSync` parameter is not specified or set to `false`, then the collection's default `waitForSync` behavior is applied. The `waitForSync` URL parameter cannot be used to disable synchronisation for collections that have a default `waitForSync` value of `true`.

The body of the response contains a JSON object with the information about the handle and the revision. The attribute `_id` contains the known `document-handle` of the updated document, the attribute `_rev` contains the new document revision.

If the document does not exist, then a `HTTP 404` is returned and the body of the response contains an error document.

There are two ways for specifying the targeted document revision id for conditional replacements (i.e. replacements that will only be executed if the revision id found in the database matches the document revision id specified in the request):

- specifying the target revision in the `rev` URL query parameter
- specifying the target revision in the `if-match` HTTP header

Specifying a target revision is optional, however, if done, only one of the described mechanisms must be used (either the `rev` URL parameter or the `if-match` HTTP header). Regardless which mechanism is used, the parameter needs to contain the target document revision id as returned in the `_rev` attribute of a document or by an HTTP `etag` header.

For example, to conditionally replace a document based on a specific revision id, you the following request:

- PUT /_api/document/`document-handle`?rev=`etag`

If a target revision id is provided in the request (e.g. via the `etag` value in the `rev` URL query parameter above), ArangoDB will check that the revision id of the document found in the database is equal to the target revision id provided in the request. If there is a mismatch between the revision id, then by default a `HTTP 412` conflict is returned and no replacement is performed.

The conditional update behavior can be overriden with the `policy` URL query parameter:

- PUT /_api/document/`document-handle`?policy=`policy`

If `policy` is set to `error`, then the behavior is as before: replacements will fail if the revision id found in the database does not match the target revision id specified in the request.

If `policy` is set to `last`, then the replacement will succeed, even if the revision id found in the database does not match the target revision id specified in the request. You can use the `last` `policy` to force replacements.

is returned if the document was created sucessfully and `waitForSync` was `true`.

is returned if the document was created sucessfully and `waitForSync` was `false`.

is returned if the body does not contain a valid JSON representation of a document. The response body contains an error document in this case.

is returned if collection or the document was not found

is returned if a "If-Match" header or `rev` is given and the found document has a different version

Using document handle:

```
unix> curl -X PUT --data @- --dump - http://localhost
   :8529/_api/document/products/35996071
{"Hello": "you"}

HTTP/1.1 202 Accepted
```

```
content-type: application/json; charset=utf-8
etag: "36323751"
location: /_api/document/products/35996071

{
  "error" : false,
  "_id" : "products/35996071",
  "_rev" : "36323751",
  "_key" : "35996071"
}
```

Unknown document handle:

```
unix> curl -X PUT --data @- --dump - http://localhost
   :8529/_api/document/products/37503399
{}

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8

{
  "error" : true,
  "errorMessage" : "document /_api/document/products
     /37503399 not found",
  "code" : 404,
  "errorNum" : 1202
}
```

Produce a revision conflict:

```
unix> curl -X PUT --header 'If-Match:"39338407"' --
   data @- --dump - http://localhost:8529/_api/
   document/products/39010727
{"other":"content"}

HTTP/1.1 412 Precondition Failed
content-type: application/json; charset=utf-8

{
  "error" : true,
  "code" : 412,
  "errorNum" : 1200,
  "errorMessage" : "precondition failed",
  "_id" : "products/39010727",
  "_rev" : "39010727",
```

```
  "_key" : "39010727"
}
```

Last write wins:

```
unix> curl -X PUT --header 'If-Match:"41042343"' --
   data @- --dump - http://localhost:8529/_api/
   document/products/40714663?policy=last
{}

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
etag: "41238951"
location: /_api/document/products/40714663

{
  "error" : false,
  "_id" : "products/40714663",
  "_rev" : "41238951",
  "_key" : "40714663"
}
```

Alternative to header field:

```
unix> curl -X PUT --data @- --dump - http://localhost
   :8529/_api/document/products/42418599?rev=42746279
{"other":"content"}

HTTP/1.1 412 Precondition Failed
content-type: application/json; charset=utf-8

{
  "error" : true,
  "code" : 412,
  "errorNum" : 1200,
  "errorMessage" : "precondition failed",
  "_id" : "products/42418599",
  "_rev" : "42418599",
  "_key" : "42418599"
}
```

PATCH /_api/document/document-handle,patches a document

document-handle (string,required) The Handle of the Document.

keepNull (string,optional) If the intention is to delete existing attributes with the patch command, the URL query parameter keepNull can be used with a value of false. This will modify the behavior of the patch command to remove any attributes from the existing document that are contained in the patch document with an attribute value of null.

waitForSync (boolean,optional) Wait until document has been sync to disk.

rev (string,optional) You can conditionally patch a document based on a target revision id by using the rev URL parameter.

policy (string,optional) To control the update behavior in case there is a revision mismatch, you can use the policy parameter.

If-Match (string,optional) You can conditionally delete a document based on a target revision id by using the if-match HTTP header.

Partially updates the document identified by document-handle. The body of the request must contain a JSON document with the attributes to patch (the patch document). All attributes from the patch document will be added to the existing document if they do not yet exist, and overwritten in the existing document if they do exist there.

Setting an attribute value to null in the patch document will cause a value of null be saved for the attribute by default.

Optionally, the URL parameter waitForSync can be used to force synchronisation of the document update operation to disk even in case that the waitForSync flag had been disabled for the entire collection. Thus, the waitForSync URL parameter can be used to force synchronisation of just specific operations. To use this, set the waitForSync parameter to true. If the waitForSync parameter is not specified or set to false, then the collection's default waitForSync behavior is applied. The waitForSync URL parameter cannot be used to disable synchronisation for collections that have a default waitForSync value of true.

The body of the response contains a JSON object with the information about the handle and the revision. The attribute _id contains the known document-handle of the updated document, the attribute _rev contains the new document revision.

If the document does not exist, then a HTTP 404 is returned and the body of the response contains an error document.

You can conditionally update a document based on a target revision id by using either the rev URL parameter or the if-match HTTP header. To control the update behavior in case there is a revision mismatch, you can use the policy parameter. This is the same as when replacing documents (see replacing documents

for details).

is returned if the document was created sucessfully and `waitForSync` was `true`.

is returned if the document was created sucessfully and `waitForSync` was `false`.

is returned if the body does not contain a valid JSON representation of a document. The response body contains an error document in this case.

is returned if collection or the document was not found

is returned if a "If-Match" header or `rev` is given and the found document has a different version

patches an existing document with new content.

```
unix> curl -X PATCH --data @- --dump - http://
   localhost:8529/_api/document/products/44122535
{"hello":"world"}

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
etag: "44450215"
location: /_api/document/products/44122535

{
  "error" : false,
  "_id" : "products/44122535",
  "_rev" : "44450215",
  "_key" : "44122535"
}

unix> curl -X PATCH --data @- --dump - http://
   localhost:8529/_api/document/products/44122535
{"numbers":{"one":1,"two":2,"three":3,"empty":null}}

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
etag: "44974503"
location: /_api/document/products/44122535

{
  "error" : false,
  "_id" : "products/44122535",
  "_rev" : "44974503",
  "_key" : "44122535"
}
```

```
unix> curl --dump - http://localhost:8529/_api/
   document/products/44122535

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: "44974503"

{
  "one" : "world",
  "hello" : "world",
  "numbers" : {
    "empty" : null,
    "one" : 1,
    "two" : 2,
    "three" : 3
  },
  "_id" : "products/44122535",
  "_rev" : "44974503",
  "_key" : "44122535"
}

unix> curl -X PATCH --data @- --dump - http://
   localhost:8529/_api/document/products/44122535?
   keepNull=false
{"hello":null,"numbers":{"four":4}}

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
etag: "45367719"
location: /_api/document/products/44122535

{
  "error" : false,
  "_id" : "products/44122535",
  "_rev" : "45367719",
  "_key" : "44122535"
}

unix> curl --dump - http://localhost:8529/_api/
   document/products/44122535

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: "45367719"
```

```
{
  "one" : "world",
  "numbers" : {
    "empty" : null,
    "one" : 1,
    "two" : 2,
    "three" : 3,
    "four" : 4
  },
  "_id" : "products/44122535",
  "_rev" : "45367719",
  "_key" : "44122535"
}
```

DELETE /_api/document/document-handle,deletes a document

document-handle (string,required) Deletes the document identified by document-handle.

rev (string,optional) You can conditionally delete a document based on a target revision id by using the rev URL parameter.

policy (string,optional) To control the update behavior in case there is a revision mismatch, you can use the policy parameter. This is the same as when replacing documents (see replacing documents for more details).

waitForSync (boolean,optional) Wait until document has been sync to disk.

If-Match (string,optional) You can conditionally delete a document based on a target revision id by using the if-match HTTP header.

The body of the response contains a JSON object with the information about the handle and the revision. The attribute _id contains the known document-handle of the updated document, the attribute _rev contains the known document revision.

If the waitForSync parameter is not specified or set to false, then the collection's default waitForSync behavior is applied. The waitForSync URL parameter cannot be used to disable synchronisation for collections that have a default waitForSync value of true.

is returned if the document was deleted sucessfully and waitForSync was true.

is returned if the document was deleted sucessfully and waitForSync was false.

is returned if the collection or the document was not found. The response body contains an error document in this case.

is returned if a "If-Match" header or rev is given and the current document has a different version

Using document handle:

```
unix> curl -X DELETE --dump - http://localhost:8529/
   _api/document/products/46678439

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8

{
  "error" : false,
  "_id" : "products/46678439",
  "_rev" : "46678439",
  "_key" : "46678439"
}
```

Unknown document handle:

```
unix> curl -X DELETE --dump - http://localhost:8529/
   _api/document/products/48054695

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8

{
  "error" : true,
  "errorMessage" : "document /_api/document/products
     /48054695 not found",
  "code" : 404,
  "errorNum" : 1202
}
```

Revision conflict:

```
unix> curl -X DELETE --header 'If-Match:"49824167"' --
   dump - http://localhost:8529/_api/document/products
   /49365415

HTTP/1.1 412 Precondition Failed
content-type: application/json; charset=utf-8

{
  "error" : true,
  "code" : 412,
  "errorNum" : 1200,
  "errorMessage" : "precondition failed",
  "_id" : "products/49365415",
  "_rev" : "49365415",
  "_key" : "49365415"
}
```

HEAD /_api/document/`document-handle`,reads a document header

document-handle (string,required) The Handle of the Document.

rev (string,optional) You can conditionally delete a document based on a target revision id by using the `rev` URL parameter.

policy (string,optional) To control the update behavior in case there is a revision mismatch, you can use the `policy` parameter. This is the same as when replacing documents (see replacing documents for more details).

If-Match (string,optional) You can conditionally get a document based on a target revision id by using the `if-match` HTTP header.

Like `GET`, but only returns the header fields and not the body. You can use this call to get the current revision of a document or check if the document was deleted.

is returned if the document was found

is returned if the document or collection was not found

is returned if the "If-None-Match" header is given and the document has same version

is returned if a "If-Match" header or `rev` is given and the found document has a different version

```
unix> curl -X HEAD --dump - http://localhost:8529/_api
   /document/products/34554279

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: "34554279"
```

reads all documents from collection

GET /_api/document,reads all documents from collection

collection (string,required) The Id of the collection.

Returns a list of all URI for all documents from the collection identified by `collection`.

All went good.

The collection does not exist.

Returns a collection.

```
unix> curl --dump - http://localhost:8529/_api/
   document/?collection=products

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8

{
  "documents" : [
    "/_api/document/products/33440167",
    "/_api/document/products/33178023",
    "/_api/document/products/32719271"
  ]
}
```

Collection does not exist.

```
unix> curl --dump - http://localhost:8529/_api/
   document/?collection=doesnotexist

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8

{
  "error" : true,
  "errorMessage" : "collection /_api/collection/
     doesnotexist not found",
  "code" : 404,
  "errorNum" : 1203
}
```

Chapter 2

REST Interface for Edges

This is an introduction to ArangoDB's REST interface for edges.

ArangoDB offers also some graph functionality. A graph consists of nodes, edges and properties. ArangoDB stores the information how the nodes relate to each other aside from the properties.

A graph data model always consists of two collections: the relations between the nodes in the graphs are stored in an "edges collection", the nodes in the graph are stored in documents in regular collections.

Example:

- the "edge" collection stores the information that a company's reception is sub-unit to the services unit and the services unit is sub-unit to the CEO. You would express this relationship with the `to` and `_to` attributes.
- the "normal" collection stores all the properties about the reception, e.g. that 20 people are working there and the room number etc.
- `_from` is the document handle of the linked vertex (incoming relation),
- `_to` is the document handle of the linked vertex (outgoing relation).

2.1 Documents, Identifiers, Handles

Edge: Edges in ArangoDB are special documents. In addition to the internal attributes `_key`, `_id` and `_rev`, they have two attributes `_from` and `_to`, which contain document handles, namely the start-point and the end-point of the edge.

`_from` and `_to` contain document handles, e.g.:

```
{ "_from" : "myvertices/doc1", "_to" : "myvertices/doc2", ... }
```

The values of _from and _to are immutable once saved.

2.2 Address and ETag of an Edge

All documents in ArangoDB have a document handle. This handle uniquely defines a document and is managed by ArangoDB. All documents are found under the URI

```
http://server:port/_api/document/document-handle
```

For edges you can use the special address

```
http://server:port/_api/edge/document-handle
```

For example: Assume that the document handle, which is stored in the _id attribute of the edge, is demo/362549736, then the URL of that edge is:

```
http://localhost:8529/_api/edge/demo/362549736
```

2.3 Working with Edges using REST

GET /_api/edge,reads an edge

GET /_api/edge/document-handle

See **REST Interface for Documents** (p. 1) for details.

POST /_api/edge,creates an edge

POST /_api/edge?collection=collection-name&from=from-handle&to=to-handle

Creates a new edge in the collection identified by collection-name. A JSON representation of the edge document must be passed as the body of the POST request. This JSON object may contain the edge's document key in the _key attribute if needed. The document handle of the start point must be passed in from-handle. The document handle of the end point must be passed in to-handle.

from-handle and to-handle are immutable once the edge has been created.

In all other respects the method works like POST /document, see **REST Interface for Documents** (p. 1) for details.

Create an edge:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/edge?collection=7848004&from
   =7848004/9289796&to=7848004/9355332
```

```
{ "e" : 1 }

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
etag: "9683012"

{
  "_rev": "9683012",
  "_id": "7848004/9683012",
  "error": false
}
```

Read an edge:

```
> curl -X GET --dump - http://localhost:8529/_api/edge
   /7848004/9683012

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: "9683012"

{
  "_from": "7848004/9289796",
  "_rev": "9683012",
  "_to": "7848004/9355332",
  "_id": "7848004/9683012",
  "e": 1
}
```

PUT /_api/edge,updates an edge

PUT /_api/edge/document-handle

See **REST Interface for Documents** (p. 1) for details.

PATCH /_api/edge,partially updates an edge

PATCH /_api/edge/document-handle

See **REST Interface for Documents** (p. 1) for details.

DELETE /_api/edge,deletes an edge

DELETE /_api/edge/document-handle

See **REST Interface for Documents** (p. 1) for details.

GET /_api/edge,reads an edge header

HEAD /_api/edge/document-handle

See **REST Interface for Documents** (p. 1) for details.

GET /_api/edges,reads in- or outbound edges

GET /_api/edges/`collection-identifier`?vertex=`vertex-handle`&direction=any

Returns the list of edges starting or ending in the vertex identified by `vertex-handle`.

GET /_api/edges/`collection-identifier`?vertex=`vertex-handle`&direction=in

Returns the list of edges ending in the vertex identified by `vertex-handle`.

GET /_api/edges/`collection-identifier`?vertex=`vertex-handle`&direction=out

Returns the list of edges starting in the vertex identified by `vertex-handle`.

Any direction

```
> curl -X GET --dump - http://localhost:8529/_api/
   edges/17501660?vertex=17501660/18419164

HTTP/1.1 200 OK
content-type: application/json

{
  "edges": [
    {
      "_from": "17501660/18419164",
      "_rev": "19140060",
      "_to": "17501660/19008988",
      "_id": "17501660/19140060"
    },
    {
      "_from": "17501660/18419164",
      "_rev": "19336668",
      "_to": "17501660/19008988",
      "_id": "17501660/19336668",
      "e": 1
    },
    {
      "_from": "17501660/19008988",
      "_rev": "19402204",
      "_to": "17501660/18419164",
      "_id": "17501660/19402204",
      "e": 2
    }
  ],
  "code": 200,
  "error": false
```

```
}
```

In edges

```
> curl -X GET --dump - http://localhost:8529/_api/
   edges/17501660?vertex=17501660/18419164&direction=
   in

HTTP/1.1 200 OK
content-type: application/json

{
  "edges": [
    {
      "_from": "17501660/19008988",
      "_rev": "19402204",
      "_to": "17501660/18419164",
      "_id": "17501660/19402204",
      "e": 2
    }
  ],
  "code": 200,
  "error": false
}
```

Out edges

```
> curl -X GET --dump - http://localhost:8529/_api/
   edges/17501660?vertex=17501660/18419164&direction=
   out

HTTP/1.1 200 OK
content-type: application/json

{
  "edges": [
    {
      "_from": "17501660/18419164",
      "_rev": "19336668",
      "_to": "17501660/19008988",
      "_id": "17501660/19336668",
      "e": 1
    },
    {
      "_from": "17501660/18419164",
```

```
      "_rev": "19140060",
      "_to": "17501660/19008988",
      "_id": "17501660/19140060"
    }
  ],
  "code": 200,
  "error": false
}
```

Chapter 3

HTTP Interface for AQL Query Cursors

3.1 Database Cursors

This is an introduction to ArangoDB's Http interface for Queries. Results of AQL and simple queries are returned as cursors in order to batch the communication between server and client. Each call returns a number of documents in a batch and an indication, if the current batch has been the final batch. Depending on the query, the total number of documents in the result set might or might not be known in advance. In order to free server resources the client should delete the cursor as soon as it is no longer needed.

To run a select query, the query details need to be shipped from the client to the server via a HTTP POST request.

3.2 Retrieving query results

Select queries are executed on-the-fly on the server and the result set will be returned back to the client.

There are two ways the client can get the result set from the server:

- in a single roundtrip
- using a cursor

3.2.1 Single roundtrip

The server will only transfer a certain number of result documents back to the client in one roundtrip. This number is controllable by the client by setting the `batchSize` attribute when issueing the query.

If the complete result can be transferred to the client in one go, the client does not need to issue any further request. The client can check whether it has retrieved the complete result set by checking the `hasMore` attribute of the result set. If it is set to `false`, then the client has fetched the complete result set from the server. In this case no server side cursor will be created.

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/cursor
{ "query" : "FOR u IN users LIMIT 2 RETURN u", "count"
    : true, "batchSize" : 2 }

HTTP/1.1 201 Created
content-type: application/json

{
  "hasMore": false,
  "error": false,
  "result": [
    {
      "n": 0,
      "_rev": "21030455",
      "_id": "19588663/21030455"
    },
    {
      "n": 1,
      "_rev": "21030455",
      "_id": "19588663/21030455"
    }
  ],
  "code": 201,
  "count": 2
}
```

3.2.2 Using a Cursor

If the result set contains more documents than should be transferred in a single roundtrip (i.e. as set via the `batchSize` attribute), the server will return the first few documents and create a temporary cursor. The cursor identifier will also be returned to the client. The server will put the cursor identifier in the `id` attribute of the response object. Furthermore, the `hasMore` attribute of the response object will be set to `true`. This is an indication for the client that there are additional results to fetch from the server.

Create and extract first batch:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/cursor
{ "query" : "FOR u IN users LIMIT 5 RETURN u", "count"
    : true, "batchSize" : 2 }

HTTP/1.1 201 Created
content-type: application/json

{
  "hasMore": true,
  "error": false,
  "id": 26011191,
  "result": [
    {
      "n": 0,
      "_rev": "25880119",
      "_id": "23914039/25880119"
    },
    {
      "n": 1,
      "_rev": "25880119",
      "_id": "23914039/25880119"
    }
  ],
  "code": 201,
  "count": 5
}
```

Extract next batch, still have more:

```
> curl -X PUT --dump - http://localhost:8529/_api/
   cursor/26011191

HTTP/1.1 200 OK
content-type: application/json

{
  "hasMore": true,
  "error": false,
  "id": 26011191,
  "result": [
    {
      "n": 2,
      "_rev": "25880119",
```

```
      "_id": "23914039/25880119"
    },
    {
      "n": 3,
      "_rev": "25880119",
      "_id": "23914039/25880119"
    }
  ],
  "code": 200,
  "count": 5
}
```

Extract next batch, done:

```
> curl -X PUT --dump - http://localhost:8529/_api/
   cursor/26011191

HTTP/1.1 200 OK
content-type: application/json

{
  "hasMore": false,
  "error": false,
  "result": [
    {
      "n": 4,
      "_rev": "25880119",
      "_id": "23914039/25880119"
    }
  ],
  "code": 200,
  "count": 5
}
```

Do not do this:

```
> curl -X PUT --dump - http://localhost:8529/_api/
   cursor/26011191

HTTP/1.1 400 Bad Request
content-type: application/json

{
  "errorNum": 1600,
  "errorMessage": "cursor not found: disposed or
```

```
    unknown cursor",
  "error": true,
  "code": 400
}
```

3.3 Accessing Cursors via HTTP

POST /_api/cursor,creates a cursor

POST /_api/cursor

The query details include the query string plus optional query options and bind parameters. These values need to be passed in a JSON representation in the body of the POST request.

The following attributes can be used inside the JSON object:

- `query`: contains the query string to be executed (mandatory)
- `count`: boolean flag that indicates whether the number of documents found should be returned as "count" attribute in the result set (optional). Calculating the "count" attribute might have a performance penalty for some queries so this option is turned off by default.
- `batchSize`: maximum number of result documents to be transferred from the server to the client in one roundtrip (optional). If this attribute is not set, a server-controlled default value will be used.
- `bindVars`: key/value list of bind parameters (optional).

If the result set can be created by the server, the server will respond with `HTTP 201`. The body of the response will contain a JSON object with the result set.

The JSON object has the following properties:

- `error`: boolean flag to indicate that an error occurred (`false` in this case)
- `code`: the HTTP status code
- `result`: an array of result documents (might be empty if query has no results)

- `hasMore`: a boolean indicator whether there are more results available on the server
- `count`: the total number of result documents available (only available if the query was executed with the `count` attribute set.
- `id`: id of temporary cursor created on the server (optional, see above)

If the JSON representation is malformed or the query specification is missing from the request, the server will respond with `HTTP 400`.

The body of the response will contain a JSON object with additional error details. The object has the following attributes:

- `error`: boolean flag to indicate that an error occurred (`true` in this case)
- `code`: the HTTP status code
- `errorNum`: the server error number
- `errorMessage`: a descriptive error message

If the query specification is complete, the server will process the query. If an error occurs during query processing, the server will respond with `HTTP 400`. Again, the body of the response will contain details about the error.

A list of query errors can be found **Error codes and meanings** (p. **??**) here.

Executes a query and extract the result in a single go:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/cursor
{ "query" : "FOR u IN users LIMIT 2 RETURN u", "count"
    : true, "batchSize" : 2 }

HTTP/1.1 201 Created
content-type: application/json

{
  "hasMore": false,
  "error": false,
  "result": [
    {
      "n": 0,
```

```
      "_rev": "21030455",
      "_id": "19588663/21030455"
    },
    {
      "n": 1,
      "_rev": "21030455",
      "_id": "19588663/21030455"
    }
  ],
  "code": 201,
  "count": 2
}
```

Bad queries:

```
> curl -X POST --dump - http://localhost:8529/_api/
   cursor

HTTP/1.1 400 Bad Request
content-type: application/json

{
  "errorNum": 1503,
  "code": 400,
  "error": true,
  "errorMessage": "query specification invalid"
}
```

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/cursor
{ "query" : "FOR u IN unknowncollection LIMIT 2 RETURN
    u.n", "count" : true, "bindVars" : {}, "batchSize"
    : 2 }

HTTP/1.1 400 Bad Request
content-type: application/json

{
  "code": 400,
  "error": true,
  "errorMessage": "unable to open collection '%s':
     unable to open collection 'unknowncollection'",
  "errorNum": 1510
}
```

POST /_api/query,parses a query

POST /_api/query

To validate a query string without executing it, the query string can be passed to the server via an HTTP POST request.

These query string needs to be passed in the attribute `query` of a JSON object as the body of the POST request.

If the query is valid, the server will respond with `HTTP 200` and return the names of the bind parameters it found in the query (if any) in the `"bindVars"` attribute of the response.

The server will respond with `HTTP 400` in case of a malformed request, or if the query contains a parse error. The body of the response will contain the error details embedded in a JSON object.

Valid query:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/query
{ "query" : "FOR u IN users FILTER u.name == @name
   LIMIT 2 RETURN u.n" }

HTTP/1.1 200 OK
content-type: application/json

{
  "error": false,
  "bindVars": [
    "name"
  ],
  "code": 200
}
```

Invalid query:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/query
{ "query" : "FOR u IN users FILTER u.name = @name
   LIMIT 2 RETURN u.n" }

HTTP/1.1 400 Bad Request
content-type: application/json

{
  "errorNum": 1501,
  "errorMessage": "parse error: %s: parse error: 1:29
```

```
    syntax error, unexpected assignment near ' =
    @name LIMIT 2 RETURN u.n'",
  "error": true,
  "code": 400
}
```

PUT /_api/cursor,reads next batch from a cursor

PUT /_api/cursor/`cursor-identifier`

If the cursor is still alive, returns an object with the following attributes.

- `id`: the `cursor-identifier`
- `result`: a list of documents for the current batch
- `hasMore`: `false` if this was the last batch
- `count`: if present the total number of elements

Note that even if `hasMore` returns `true`, the next call might still return no documents. If, however, `hasMore` is `false`, then the cursor is exhausted. Once the `hasMore` attribute has a value of `false`, the client can stop.

The server will respond with `HTTP 200` in case of success. If the cursor identifier is ommitted or somehow invalid, the server will respond with `HTTP 404`.

Valid request for next batch:

```
> curl -X PUT --dump - http://localhost:8529/_api/
   cursor/26011191

HTTP/1.1 200 OK
content-type: application/json

{
  "hasMore": true,
  "error": false,
  "id": 26011191,
  "result": [
    {
      "n": 2,
      "_rev": "25880119",
      "_id": "23914039/25880119"
    },
    {
      "n": 3,
      "_rev": "25880119",
      "_id": "23914039/25880119"
    }
  ],
  "code": 200,
  "count": 5
}
```

Missing identifier

```
> curl -X PUT --dump - http://localhost:8529/_api/
   cursor

HTTP/1.1 400 Bad Request
content-type: application/json

{
  "code": 400,
  "errorMessage": "bad parameter",
  "errorNum": 400,
  "error": true
}
```

Unknown identifier

```
> curl -X PUT --dump - http://localhost:8529/_api/
   cursor/123456

HTTP/1.1 400 Bad Request
content-type: application/json

{
  "code": 400,
  "errorNum": 1600,
  "error": true,
  "errorMessage": "cursor not found: disposed or
     unknown cursor"
}
```

DELETE /_api/cursor,deletes a cursor

DELETE /_api/cursor/`cursor-identifier`

Deletes the cursor and frees the resources associated with it.

The cursor will automatically be destroyed on the server when the client has retrieved all documents from it. The client can also explicitly destroy the cursor at any earlier time using an HTTP DELETE request. The cursor id must be included as part of the URL.

In case the server is aware of the cursor, it will respond with `HTTP 202`. Otherwise, it will respond with `404`.

Cursors that have been explicitly destroyed must not be used afterwards. If a cursor is used after it has been destroyed, the server will respond with `HTTP 404` as well.

Note: the server will also destroy abandoned cursors automatically after a certain server-controlled timeout to avoid resource leakage.

```
> curl -X DELETE --dump - http://localhost:8529/_api/
   cursor/8679702

HTTP/1.1 202 Accepted
content-type: application/json

{
  "code": 202,
  "id": "8679702",
  "error": false
}
```

Chapter 4

HTTP Interface for AQL Queries

ArangoDB has an Http interface to syntactically validate AQL queries. Furthermore, it offers an Http interface to retrieve the execution plan for any valid AQL query.

Both functionalities do not actually execute the supplied AQL query, but only inspect it and return meta information about it.

POST /_api/explain,explains a query

POST /_api/explain

To explain how an AQL query would be executed on the server, the query string can be sent to the server via an HTTP POST request. The server will then validate the query and create an execution plan for it, but will not execute it.

The execution plan that is returned by the server can be used to estimate the probable performance of an AQL query. Though the actual performance will depend on many different factors, the execution plan normally can give some good hint on the amount of work the server needs to do in order to actually run the query.

The query string needs to be passed in the attribute `query` of a JSON object as the body of the POST request. If the query references any bind variables, these must also be passed in the attribute `bindVars`.

If the query is valid, the server will respond with `HTTP 200` and return a list of the individual query execution steps in the `"plan"` attribute of the response.

The server will respond with `HTTP 400` in case of a malformed request, or if the query contains a parse error. The body of the response will contain the error details embedded in a JSON object. Omitting bind variables if the query references any will result also result in an `HTTP 400` error.

Valid query:

```
> curl --data @- -X POST --dump - http://localhost
```

```
   :8529/_api/explain
{ "query" : "FOR u IN users FILTER u.id == @id LIMIT 2
    RETURN u.name", "bindVars": { "id" : 3 } }

HTTP/1.1 200 OK
content-type: application/json

{
  "plan": [
    {
      "id" : 1,
      "loopLevel" : 1,
      "type" : "for",
      "resultVariable" : "u",
      "expression" : {
        "type" : "collection",
        "value" : "users",
        "extra" : {
          "accessType" : "all"
        }
      }
    },
    {
      "id" : 2,
      "loopLevel" : 1,
      "type" : "filter",
      "expression" : {
        "type" : "expression",
        "value" : "u.id == 3"
      }
    },
    {
      "id" : 3,
      "loopLevel" : 1,
      "type" : "limit",
      "offset" : 0,
      "count" : 2
    },
    {
      "id" : 4,
      "loopLevel" : 1,
      "type" : "limit",
      "offset" : 0,
      "count" : 2
```

```
    },
    {
      "id" : 5,
      "loopLevel" : 1,
      "type" : "return",
      "expression" : {
        "type" : "expression",
        "value" : "u.name"
      }
    }
  ],
  "error": false,
  "code": 200
}
```

Invalid query:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/explain
{ "query" : "FOR u IN users FILTER u.name == @name
   LIMIT 2 RETURN u.n" }

HTTP/1.1 400 Bad Request
content-type: application/json

{
  "errorNum": 1551,
  "errorMessage": "no value specified for declared
     bind parameter 'name'",
  "error": true,
  "code": 400
}
```

The data returned in the `plan` attribute of the result contains one element per AQL top-level statement (i.e. `FOR`, `RETURN`, `FILTER` etc.). If the query optimiser removed some unnecessary statements, the result might also contain less elements than there were top-level statements in the AQL query. The following example shows a query with a non-sensible filter condition that the optimiser has removed so that there are less top-level statements:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/explain
{ "query" : "FOR i IN [ 1, 2, 3 ] FILTER 1 == 2 RETURN
    i" }
```

```
HTTP/1.1 200 OK
content-type: application/json

{
  "plan": [
    {
      "id" : 1,
      "loopLevel" : 0,
      "type" : "return (empty)",
      "expression" : {
        "type" : "const list",
        "value" : "[  ]"
      }
    }
  ],
  "error": false,
  "code": 200
}
```

The top-level statements will appear in the result in the same order in which they have been used in the original query. Each result element has at most the following attributes:

- `id`: the row number of the top-level statement, starting at 1
- `type`: the type of the top-level statement (e.g. `for`, `return` ...)
- `loopLevel`: the nesting level of the top-level statement, starting at 1 Depending on the type of top-level statement, there might be other attributes providing additional information, for example, if and which indexed will be used. Many top-level statements will provide an `expression` attribute that contains data about the expression they operate on. This is true for `FOR`, `FILTER`, `SORT`, `COLLECT`, and `RETURN` statements. The `expression` attribute has the following sub-attributes:
- `type`: the type of the expression. Some possible values are:
  - `collection`: an iteration over documents from a collection. The `value` attribute will then contain the collection name. The `extra` attribute will contain information about if and which index is used when accessing the documents from the collection. If no index is used, the `accessType` sub-attribute of the `extra` attribute will have the value `all`, otherwise it will be `index`.
  - `list`: a list of dynamic values. The `value` attribute will contain the list elements.

- `const list`: a list of constant values. The `value` attribute will contain the list elements.
- `reference`: a reference to another variable. The `value` attribute will contain the name of the variable that is referenced.

Please note that the structure of the explain result data might change in future versions of ArangoDB without further notice and without maintaining backwards compatibility.

POST /_api/query,parses a query

POST /_api/query

To validate a query string without executing it, the query string can be passed to the server via an HTTP POST request.

These query string needs to be passed in the attribute `query` of a JSON object as the body of the POST request.

If the query is valid, the server will respond with `HTTP 200` and return the names of the bind parameters it found in the query (if any) in the `"bindVars"` attribute of the response.

The server will respond with `HTTP 400` in case of a malformed request, or if the query contains a parse error. The body of the response will contain the error details embedded in a JSON object.

Valid query:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/query
{ "query" : "FOR u IN users FILTER u.name == @name
   LIMIT 2 RETURN u.n" }

HTTP/1.1 200 OK
content-type: application/json

{
  "error": false,
  "bindVars": [
    "name"
  ],
  "code": 200
}
```

Invalid query:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/query
{ "query" : "FOR u IN users FILTER u.name = @name
   LIMIT 2 RETURN u.n" }

HTTP/1.1 400 Bad Request
content-type: application/json

{
  "errorNum": 1501,
  "errorMessage": "parse error: %s: parse error: 1:29
```

```
    syntax error, unexpected assignment near ' =
    @name LIMIT 2 RETURN u.n'",
  "error": true,
  "code": 400
}
```

Chapter 5

HTTP Interface for AQL User Functions Management

5.1 AQL User Functions Management

This is an introduction to ArangoDB's Http interface for managing AQL user functions. AQL user functions are a means to extend the functionality of ArangoDB's query language (AQL) with user-defined Javascript code.

For an overview of how AQL user functions work, please refer to **Extending AQL with User Functions** (p. **??**).

The Http interface provides an API for adding, deleting, and listing previously registered AQL user functions.

All user functions managed through this interface will be stored in the system collection `_aqlfunctions`. Documents in this collection should not be accessed directly, but only via the dedicated interfaces.

POST /_api/aqlfunction,creates or replaces an AQL user function

POST /_api/aqlfunction

The following data need to be passed in a JSON representation in the body of the POST request:

- `name`: the fully qualified name of the user functions.
- `code`: a string representation of the function body.
- `isDeterministic`: an optional boolean value to indicate that the function results are fully deterministic (function return value solely depends on

the input value and return value is the same for repeated calls with same input). The `isDeterministic` attribute is currently not used but may be used later for optimisations.

If the function can be registered by the server, the server will respond with `HTTP 201`. If the function already existed and was replaced by the call, the server will respond with `HTTP 200`.

In case of success, the returned JSON object has the following properties:

- `error`: boolean flag to indicate that an error occurred (`false` in this case)
- `code`: the HTTP status code

If the JSON representation is malformed or mandatory data is missing from the request, the server will respond with `HTTP 400`.

The body of the response will contain a JSON object with additional error details. The object has the following attributes:

- `error`: boolean flag to indicate that an error occurred (`true` in this case)
- `code`: the HTTP status code
- `errorNum`: the server error number
- `errorMessage`: a descriptive error message

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/aqlfunction
{ "name" : "myfunctions:temperature:
   celsiustofahrenheit, "code" : "function (celsius) {
    return celsius * 1.8 + 32; }" }

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8

{
  "error" : false,
  "code" : 201
}
```

DELETE /_api/aqlfunction,remove an existing AQL user function

DELETE /_api/aqlfunction/`name`

group (string,optional) If set to `true`, then the function name provided in `name` is treated as a namespace prefix, and all functions in the specified namespace will be deleted.

If set to `false`, the function name provided in `name` must be fully qualified, including any namespaces.

Removes an existing AQL user function, identified by `name`.

If the function can be removed by the server, the server will respond with `HTTP 200`.

In case of success, the returned JSON object has the following properties:

- `error`: boolean flag to indicate that an error occurred (`false` in this case)
- `code`: the HTTP status code

If the JSON representation is malformed or mandatory data is missing from the request, the server will respond with `HTTP 400`. If the specified user does not exist, the server will respond with `HTTP 404`.

The body of the response will contain a JSON object with additional error details. The object has the following attributes:

- `error`: boolean flag to indicate that an error occurred (`true` in this case)
- `code`: the HTTP status code
- `errorNum`: the server error number
- `errorMessage`: a descriptive error message

GET /_api/aqlfunction,returns registered AQL user functions

GET /_api/aqlfunction

Returns all registered AQL user functions.

GET /_api/aqlfunction?namespace=`namespace`

Returns all registered AQL user functions from namespace `namespace`.

The call will return a JSON list with all user functions found. Each user function will at least have the following attributes:

- `name`: The fully qualified name of the user function
- `code`: A string representation of the function body

Chapter 6

HTTP Interface for Simple Queries

6.1 Simple Queries

This is an introduction to ArangoDB's Http interface for simple queries.

Simple queries can be used if the query condition is straight forward simple, i.e., a document reference, all documents, a query-by-example, or a simple geo query. In a simple query you can specify exactly one collection and one condition. The result can then be sorted and can be split into pages.

6.2 Working with Simples Queries using HTTP

To limit the amount of results to be transferred in one batch, simple queries support a `batchSize` parameter that can optionally be used to tell the server to limit the number of results to be transferred in one batch to a certain value. If the query has more results than were transferred in one go, more results are waiting on the server so they can be fetched subsequently. If no value for the `batchSize` parameter is specified, the server will use a reasonable default value.

If the server has more documents than should be returned in a single batch, the server will set the `hasMore` attribute in the result. It will also return the id of the server-side cursor in the `id` attribute in the result. This id can be used with the cursor API to fetch any outstanding results from the server and dispose the server-side cursor afterwards.

PUT /_api/simple/all,executes simple query "all"

PUT /_api/simple/all

Returns all documents of a collections. The call expects a JSON object as body with the following attributes:

- `collection`: The name of the collection to query.
- `skip`: The number of documents to skip in the query (optional).
- `limit`: The maximal amount of documents to return. The `skip` is applied before the `limit` restriction. (optional)

Returns a cursor containing the result, see **HTTP Interface for AQL Query Cursors** (p. 28) for details.

Limit the amount of documents using `limit`

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/all
{ "collection" : "4421689145", "skip" : 2900, "limit"
   : 2 }

HTTP/1.1 201 Created
content-type: application/json

{
  "code": 201,
  "hasMore": false,
  "result": [
    {
      "n": 1679,
      "_rev": "4533165881",
      "_id": "4421689145/4533165881"
    },
    {
      "n": 679,
      "_rev": "4467629881",
      "_id": "4421689145/4467629881"
    }
  ],
  "count": 2,
  "error": false
}
```

Using a `batchSize` value

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/all
{ "collection" : "4421689145", "batchSize" : 2 }
```

```
HTTP/1.1 201 Created
content-type: application/json

{
  "code": 201,
  "hasMore": true,
  "result": [
    {
      "n": 1679,
      "_rev": "4533165881",
      "_id": "4421689145/4533165881"
    },
    {
      "n": 679,
      "_rev": "4467629881",
      "_id": "4421689145/4467629881"
    }
  ],
  "id":142116766,
  "count": 3000,
  "error": false
}
```

PUT /_api/simple/by-example,executes simple query "by-example"

PUT /_api/simple/by-example

This will find all documents matching a given example.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `example`: The example.
- `skip`: The documents to skip in the query. (optional)
- `limit`: The maximal amount of documents to return. (optional)

Returns a cursor containing the result, see **HTTP Interface for AQL Query Cursors** (p. 28) for details.

Matching an attribute:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/by-example
{ "collection" : "3179705695", "example" : [ { "i" : 1
    } ] }

HTTP/1.1 201 Created
content-type: application/json

{
  "result": [
    { "a": { "k": 2, "j": 2 }, "i": 1, "_rev":
       "3181802847", "_id": "3179705695/3181802847" },
    { "a": { "j": 1 }, "i": 1, "_rev": "3181475167", "
       _id": "3179705695/3181475167" },
    { "a": { "k": 1, "j": 1 }, "i": 1, "_rev":
       "3181737311", "_id": "3179705695/3181737311" },
    { "i": 1, "_rev": "3181147487", "_id":
       "3179705695/3181147487" }
  ],
  "count": 4,
  "error": false,
  "hasMore": false,
  "code": 201
}
```

Matching an attribute which is a sub-document:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/by-example
{ "collection" : "3179705695", "example" : [ { "a" : {
    "j" : 1 } } ] }

HTTP/1.1 201 Created
content-type: application/json

{
  "result": [
    { "a": { "j": 1 }, "i": 1, "_rev": "3181475167", "
       _id": "3179705695/3181475167" }
  ],
  "count": 1,
  "error": false,
  "hasMore": false,
  "code": 201
}
```

Matching an attribute within a sub-document:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/by-example
{ "collection" : "3179705695", "example" : [ "a.j", 1
   ] }

HTTP/1.1 201 Created
content-type: application/json

{
  "result": [
    { "a": { "j": 1 }, "i": 1, "_rev": "3181475167", "
       _id": "3179705695/3181475167" },
    { "a": { "k": 1, "j": 1 }, "i": 1, "_rev":
       "3181737311", "_id": "3179705695/3181737311" }
  ],
  "count": 2,
  "error": false,
  "hasMore": false,
  "code": 201
}
```

PUT /_api/simple/first-example,executes simple query "first-example"

PUT /_api/simple/first-example

This will return the first document matching a given example.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `example`: The example.

Returns a result containing the document or `HTTP 404` if no document matched the example.

If a matching document was found:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/first-example
{ "collection" : "666351134", "example" : [ "a.j", 1,
   "a.k", 1 ] }

HTTP/1.1 200 OK
content-type: application/json

{
  "error": false,
  "code": 200,
  "document": { "_rev": "668382750", "_id":
     "666351134/668382750", "a": { "k": 1, "j": 1, "l"
      : 10 }, "i": 1 }
}
```

If no document was found:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/first-example
{ "collection" : "666351134", "example" : [ "a.j", 1,
   "a.k", 2 ] }

HTTP/1.1 404 Not Found
content-type: application/json

{
  "errorMessage": "no match",
  "error": true,
  "code": 404,
```

```
  "errorNum": 404
}
```

PUT /_api/simple/any,executes simple query "any"

PUT /_api/simple/any

Returns a random document of a collection. The call expects a JSON object as body with the following attributes:

- `collection`: The identifier or name of the collection to query.

Returns a JSON object with the document stored in the attribute `document` if the collection contains at least one document. If the collection is empty, the attrbute contains null.

```
> curl --data @- -X PUT --dump - http://localhost:8529/_api/simple/any
{ "collection" : 222186062247 }

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
content-length: 114

{
  "document": {
    "_id": "222186062247/223172116903",
    "_rev": 223172116903,
    "Hello": "World"
  },
  "error": false,
  "code":200
}
```

PUT /_api/simple/range,executes simple range query

PUT /_api/simple/range

This will find all documents within a given range. You must declare a skip-list index on the attribute in order to be able to use a range query.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `attribute`: The attribute path to check.
- `left`: The lower bound.
- `right`: The upper bound.
- `closed`: If true, use interval including `left` and `right`, otherwise exclude `right`, but include `left`.
- `skip`: The documents to skip in the query. (optional)
- `limit`: The maximal amount of documents to return. (optional)

Returns a cursor containing the result, see **HTTP Interface for AQL Query Cursors** (p. 28) for details.

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/range
{ "collection" : "6328369086", "attribute" : "i", "
   left" : 2, "right" : 4 }

HTTP/1.1 201 Created
content-type: application/json

{
  "code": 201,
  "result": [
    { "_id": "6328369086/6329876414", "i": 2, "_rev":
       "6329876414" },
    { "_id": "6328369086/6329941950", "i": 3, "_rev":
       "6329941950" }
  ],
  "count": 2,
```

```
  "hasMore": false,
  "error": false
}
```

PUT /_api/simple/near,executes simple query "near"

PUT /_api/simple/near

The default will find at most 100 documents near a given coordinate. The returned list is sorted according to the distance, with the nearest document coming first. If there are near documents of equal distance, documents are chosen randomly from this set until the limit is reached.

In order to use the `near` operator, a geo index must be defined for the collection. This index also defines which attribute holds the coordinates for the document. If you have more then one geo-spatial index, you can use the `geo` field to select a particular index.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `latitude`: The latitude of the coordinate.
- `longitude`: The longitude of the coordinate.
- `distance`: If given, the attribute key used to store the distance. (optional)
- `skip`: The number of documents to skip in the query. (optional)
- `limit`: The maximal amount of documents to return. The `skip` is applied before the `limit` restriction. The default is 100. (optional)
- `geo`: If given, the identifier of the geo-index to use. (optional)

Returns a cursor containing the result, see **HTTP Interface for AQL Query Cursors** (p. 28) for details.

Without distance:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/near
{ "collection" : "11172897562", "latitude" : 0, "
   longitude" : 0, "skip" : 1, "limit" : 2 }

HTTP/1.1 201 Created
content-type: application/json

{
```

```
  "hasMore": false,
  "count": 5,
  "result": [
    {
      "_id": "11172897562/11178271514",
      "loc": [ 0, -10 ],
      "_rev": "11178271514"
    },
    {
      "_id": "11172897562/11177616154",
      "loc": [ -10, 0 ],
      "_rev": "11177616154"
    }
  ],
  "code": 201,
  "error": false
}
```

With distance:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/near
{ "collection" : "11182465818", "latitude" : 0, "
   longitude" : 0, "limit" : 2, "distance" : "distance
   " }

HTTP/1.1 201 Created
content-type: application/json

{
  "hasMore": false,
  "count": 5,
  "result": [
    {
      "distance": 0,
      "_id": "11182465818/11187905306",
      "loc": [ 0, 0 ],
      "_rev": "11187905306"
    },
    {
      "distance": 1111949.26644559,
      "_id": "11182465818/11187839770",
      "loc": [ 0, -10 ],
      "_rev": "11187839770"
```

```
    }
  ],
  "code": 201,
  "error": false
}
```

PUT /_api/simple/within,executes simple query "within"

PUT /_api/simple/within

This will find all documents with in a given radius around the coordinate (`latitude`, `longitude`). The returned list is sorted by distance.

In order to use the `within` operator, a geo index must be defined for the collection. This index also defines which attribute holds the coordinates for the document. If you have more then one geo-spatial index, you can use the `geo` field to select a particular index.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `latitude`: The latitude of the coordinate.
- `longitude`: The longitude of the coordinate.
- `radius`: The maximal radius (in meters).
- `distance`: If given, the result attribute key used to store the distance values (optional). If specified, distances are returned in meters.
- `skip`: The documents to skip in the query. (optional)
- `limit`: The maximal amount of documents to return. (optional)
- `geo`: If given, the identifier of the geo-index to use. (optional)

Returns a cursor containing the result, see **HTTP Interface for AQL Query Cursors** (p. 28) for details.

Without distance:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/within
{ "collection" : "2076584128", "latitude" : 0, "
   longitude" : 0, "skip" : 1, "radius" : 1111950 }

HTTP/1.1 201 Created
content-type: application/json

{
```

```
  "result": [
    {
      "_id": "2076584128/2082744512",
      "loc": [ 10, 0 ],
      "_rev": "2082744512"
    },
    {
      "_id": "2076584128/2082089152",
      "loc": [ 0, 10 ],
      "_rev": "2082089152"
    },
    {
      "_id": "2076584128/2081302720",
      "loc": [ -10, 0 ],
      "_rev": "2081302720"
    },
    {
      "_id": "2076584128/2081958080",
      "loc": [ 0, -10 ],
      "_rev": "2081958080"
    }
  ],
  "code": 201,
  "hasMore": false,
  "count": 4,
  "error": false
}
```

With distance:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/within
{ "collection" : "2086152384", "latitude" : 0, "
   longitude" : 0, "distance" : "distance", "radius" :
    1111950 }

HTTP/1.1 201 Created
content-type: application/json

{
  "result": [
    {
      "distance": 0,
      "_id": "2086152384/2091591872",
```

```
      "loc": [ 0, 0 ],
      "_rev": "2091591872"
    },
    {
      "distance": 1111949.26644559,
      "_id": "2086152384/2090870976",
      "loc": [ -10, 0 ],
      "_rev": "2090870976"
    },
    {
      "distance": 1111949.26644559,
      "_id": "2086152384/2091657408",
      "loc": [ 0, 10 ],
      "_rev": "2091657408"
    },
    {
      "distance": 1111949.26644559,
      "_id": "2086152384/2092312768",
      "loc": [ 10, 0 ],
      "_rev": "2092312768"
    },
    {
      "distance": 1111949.26644559,
      "_id": "2086152384/2091526336",
      "loc": [ 0, -10 ],
      "_rev": "2091526336"
    }
  ],
  "code": 201,
  "hasMore": false,
  "count": 5,
  "error": false
}
```

PUT /_api/simple/fulltext,executes simple query "fulltext"

PUT /_api/simple/fulltext

This will find all documents from the collection that match the fulltext query specified in `query`.

In order to use the `fulltext` operator, a fulltext index must be defined for the collection and the specified attribute.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `attribute`: The attribute that contains the texts.
- `query`: The fulltext query.
- `skip`: The documents to skip in the query. (optional)
- `limit`: The maximal amount of documents to return. (optional)
- `index`: If given, the identifier of the fulltext-index to use. (optional)

Returns a cursor containing the result, see **HTTP Interface for AQL Query Cursors** (p. 28) for details.

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/fulltext
{ "collection" : "2076584128", "attribute" : "text", "
   query" : "word" }

HTTP/1.1 201 Created
content-type: application/json

{
  "result": [
    {
      "_id": "2076584128/2082744512",
      "text" : "this text contains a word",
      "_rev": "2082744512"
    },
    {
      "_id": "2076584128/2082089152",
      "text" : "this text also contains a word",
```

```
      "loc": [ 0, 10 ],
      "_rev": "2082089152"
    }
  ],
  "code": 201,
  "hasMore": false,
  "count": 2,
  "error": false
}
```

PUT /_api/simple/remove-by-example,removes documents by example

PUT /_api/simple/remove-by-example

This will find all documents in the collection that match the specified example object.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to remove from.
- `example`: An example object that all collection objects are compared against.
- `waitForSync`: if set to true, then all removal operations will instantly be synchronised to disk. If this is not specified, then the collection's default sync behavior will be applied.
- `limit`: an optional value that determines how many documents to delete at most. If `limit` is specified but is less than the number of documents in the collection, it is undefined which of the documents will be deleted.

Returns the number of documents that were deleted

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/remove-by-example
{ "collection" : "test", "example" : { "age" : 37, "
   likes" : "tennis" } }

HTTP/1.1 200 Ok
content-type: application/json

{
  "code": 200,
  "deleted": 4,
  "error": false
}
```

PUT /_api/simple/replace-by-example,replaces documents by example

PUT /_api/simple/replace-by-example

This will find all documents in the collection that match the specified example object, and replace the entire document body with the new value specified. Note that document meta-attributes such as `_id`, `_key`, `_from`, `_to` etc. cannot be replaced.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to replace within.
- `example`: An example object that all collection objects are compared against.
- `newValue`: The replacement document that will get inserted in place of the "old" documents.
- `waitForSync`: if set to true, then all removal operations will instantly be synchronised to disk. If this is not specified, then the collection's default sync behavior will be applied.
- `limit`: an optional value that determines how many documents to replace at most. If `limit` is specified but is less than the number of documents in the collection, it is undefined which of the documents will be replaced.

Returns the number of documents that were replaced

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/replace-by-example
{ "collection" : "test", "example" : { "age" : 37, "
   likes" : "tennis" }, "newValue" : { "foo" : "bar"
   }, "limit" : 3 }

HTTP/1.1 200 Ok
content-type: application/json

{
  "code": 200,
  "replaced": 1,
  "error": false
}
```

PUT /_api/simple/update-by-example,updates documents by example

PUT /_api/simple/update-by-example

This will find all documents in the collection that match the specified example object, and partially update the document body with the new value specified. Note that document meta-attributes such as `_id`, `_key`, `_from`, `_to` etc. cannot be replaced.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to update within.
- `example`: An example object that all collection objects are compared against.
- `newValue`: The update value that will get inserted in place of the "old" version of the found documents.
- `keepNull`: This parameter can be used to modify the behavior when handling `null` values. Normally, `null` values are stored in the database. By setting the `keepNull` parameter to `false`, this behavior can be changed so that all attributes in `data` with `null` values will be removed from the updated document.
- `waitForSync`: if set to true, then all removal operations will instantly be synchronised to disk. If this is not specified, then the collection's default sync behavior will be applied.
- `limit`: an optional value that determines how many documents to update at most. If `limit` is specified but is less than the number of documents in the collection, it is undefined which of the documents will be updated.

Returns the number of documents that were updated

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/update-by-example
{ "collection" : "test", "example" : { "age" : 37, "
   likes" : "tennis" }, "newValue" : { "age" : null, "
   likes" : "foo" }, "keepNull" : false }

HTTP/1.1 200 Ok
content-type: application/json

{
```

```
  "code": 200,
  "updated": 1,
  "error": false
}
```

Chapter 7

HTTP Interface for Collections

7.1 Collections

This is an introduction to ArangoDB's Http interface for collections.

Collection: A collection consists of documents. It is uniquely identified by the server via its collection identifier. It also has a unique name that clients should use to identify and access it. Collections have a type that is specified by the user when the collection is created. There currently are *document* and *edge* collections. The default type is *document*.

Collection Identifier: A collection identifier identifies a collection in a database. It is a string value and is unique within the database. Up to including ArangoDB 1.1, the collection identifier has been a client's primary means to access collections. Starting with ArangoDB 1.2, clients should instead use a collection's unique name to access a collection instead of its identifier.

ArangoDB currently uses 64bit unsigned integer values to maintain collection ids internally. When returning collection ids to clients, ArangoDB will put them into a string to ensure the collection id is not clipped by clients that do not support big integers. Clients should treat the collection ids returned by ArangoDB as opaque strings when they store or use it locally.

Note: collection ids have been returned as integers up to including ArangoDB 1.1

Collection Name: A collection name identifies a collection in a database. It is a string and is unique within the database. Unlike the collection identifier it is supplied by the creator of the collection. The collection name must consist of letters, digits and the characters _ (underscore), - (dash), and : (colon). Please refer to **Naming Conventions in ArangoDB** (p. 190) for more information on valid collection names.

Key Generator: ArangoDB allows using key generators for each collection. Key

generators have the purpose of auto-generating values for the _key attribute of a document if none was specified by the user.

By default, ArangoDB will use the traditional key generator. The traditional key generator will auto-generate key values that are strings with ever-increasing numbers. The increment values it uses are non-deterministic.

Contrary, the autoincrement key generator will auto-generate deterministic key values. Both the start value and the increment value can be defined when the collection is created. The default start value is 0 and the default increment is 1, meaning the key values it will create by default are:

```
1, 2, 3, 4, 5, ...
```

When creating a collection with the autoincrement key generator and an increment of 5, the generated keys would be:

```
1, 6, 11, 16, 21, ...
```

The basic operations (create, read, update, delete) for documents are mapped to the standard HTTP methods (POST, GET, PUT, DELETE).

7.2 Address of a Collection

All collections in ArangoDB have an unique identifier and a unique name. ArangoDB internally uses the collection's unique identifier to look up collections. This identifier however is managed by ArangoDB and the user has no control over it. In order to allow users use their own names, each collection also has a unique name, which is specified by the user. To access a collection from the user perspective, the collection name should be used, i.e.:

```
http://server:port/_api/collection/collection-name
```

For example: Assume that the collection identifier is 7254820 and the collection name is demo, then the URL of that collection is:

```
http://localhost:8529/_api/collection/demo
```

7.3 Working with Collections using HTTP

7.3.1 Creating and Deleting Collections

POST /_api/collection,creates a collection

Creates an new collection with a given name. The request must contain an object with the following attributes.

- `name`: The name of the collection.

- `waitForSync` (optional, default: false): If `true` then the data is synchronised to disk before returning from a create or update of an document.

- `journalSize` (optional, default is a **configuration parameter** (p. **??**))-: The maximal size of a journal or datafile. Note that this also limits the maximal size of a single object. Must be at least 1MB.

- `isSystem` (optional, default is `false`): If `true`, create a system collection. In this case `collection-name` should start with an underscore. End users should normally create non-system collections only. API implementors may be required to create system collections in very special occasions, but normally a regular collection will do.

- `isVolatile` (optional, default is `false`): If `true` then the collection data is kept in-memory only and not made persistent. Unloading the collection will cause the collection data to be discarded. Stopping or re-starting the server will also cause full loss of data in the collection. Setting this option will make the resulting collection be slightly faster than regular collections because ArangoDB does not enforce any synchronisation to disk and does not calculate any CRC checksums for datafiles (as there are no datafiles).

  This option should threrefore be used for cache-type collections only, and not for data that cannot be re-created otherwise.

- `keyOptions` (optional) additional options for key generation. If specified, then `keyOptions` should be a JSON array containing the following attributes (note: some of them are optional):

  - `type`: specifies the type of the key generator. The currently available generators are `traditional` and `autoincrement`.
  - `allowUserKeys`: if set to `true`, then it is allowed to supply own key values in the `_key` attribute of a document. If set to `false`, then the key generator will solely be responsible for generating keys and supplying own key values in the `_key` attribute of documents is considered an error.
  - `increment`: increment value for `autoincrement` key generator. Not used for other key generator types.
  - `offset`: initial offset value for `autoincrement` key generator. Not used for other key generator types.

- `type` (optional, default is `2`): the type of the collection to create. The following values for `type` are valid:

- 2: document collection
- 3: edges collection

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/collection
{ "name" : "UnitTestsCollectionBasics" }

HTTP/1.1 200 OK
content-type: application/json

{
  "name": "UnitTestsCollectionBasics",
  "code": 200,
  "waitForSync": false,
  "isVolatile": false,
  "id": 179665369,
  "status": 3,
  "type" : 2,
  "error": false
}

> curl --data @- -X POST --dump - http://localhost
   :8529/_api/collection
{ "name" : "UnitTestsCollectionEdges", "type" : 3 }

HTTP/1.1 200 OK
content-type: application/json

{
  "name": "UnitTestsCollectionEdges",
  "code": 200,
  "waitForSync": false,
  "isVolatile": false,
  "id": 184354432,
  "status": 3,
  "type": 3,
  "error": false
}
```

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/collection
{ "name" : "users", "keyOptions" : { "type" : "
   autoincrement", "increment" : 5, "allowUserKeys" :
   true } }
```

```
HTTP/1.1 200 OK
location: /_api/collection/users
content-type: application/json; charset=utf-8

{
  "name" : "users",
  "waitForSync" : false,
  "isVolatile" : false,
  "isSystem" : false,
  "status" : 3,
  "type" : 2,
  "error" : false,
  "code": 200
}
```

DELETE /_api/collection/{collection-name},deletes a collection

collection-name (string,required)

Deletes a collection identified by `collection-name`.

If the collection was successfully deleted then, an object is returned with the following attributes:

- `error`: `false`
- `id`: The identifier of the deleted collection.

If the `collection-name` is missing, then a `HTTP 400` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

Using an identifier:

```
> curl -X DELETE --dump - http://localhost:8529/_api/
   collection/101711425

HTTP/1.1 200 OK
content-type: application/json

{
  "code": 200,
  "id": 101711425,
  "error": false
}
```

Using a name:

```
> curl -X DELETE --dump - http://localhost:8529/_api/
   collection/UnitTestsCollectionBasics

HTTP/1.1 200 OK
content-type: application/json

{
  "code": 200,
  "id": 103022145,
  "error": false
}
```

PUT /_api/collection/{collection-name}/truncate,truncates a collection

collection-name (string,required)

Removes all documents from the collection, but leaves the indexes intact.

```
> curl -X PUT --dump - http://localhost:8529/_api/
   collection/56478340/truncate

HTTP/1.1 200 OK
content-type: application/json

{
  "name": "UnitTestsCollectionBasics",
  "code": 200,
  "id": 56478340,
  "status": 3,
  "type": 2,
  "error": false
}
```

7.3.2 Getting Information about a Collection

GET /_api/collection/{collection-name},reads a collection

collection-name (string,required)

The result is an object describing the collection with the following attributes:

- `id`: The identifier of the collection.
- `name`: The name of the collection.
- `status`: The status of the collection as number.
  - 1: new born collection
  - 2: unloaded
  - 3: loaded
  - 4: in the process of being unloaded
  - 5: deleted

Every other status indicates a corrupted collection.

- `type`: The type of the collection as number.
  - 2: document collection (normal case)
  - 3: edges collection

If the `collection-name` is unknown, then a `HTTP 404` is returned.

GET /_api/collection/{collection-name}/properties,reads a collection with properties

collection-name (string,required)

In addition to the above, the result will always contain the `waitForSync`, `journalSize`, and `isVolatile` properties. This is achieved by forcing a load of the underlying collection.

- `waitForSync`: If `true` then creating or changing a document will wait until the data has been synchronised to disk.
- `journalSize`: The maximal size of a journal / datafile.
- `isVolatile`: If `true` then the collection data will be kept in memory only and ArangoDB will not write or sync the data to disk.

If the `collection-name` is missing, then a `HTTP 400` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

Using an identifier:

```
> curl -X GET --dump - http://localhost:8529/_api/
   collection/73482

HTTP/1.1 200 OK
content-type: application/json

{
  "name": "UnitTestsCollectionBasics",
  "code": 200,
  "id": 73482,
  "status": 3,
  "type": 2,
  "error": false
}
```

Using a name:

```
> curl -X GET --dump - http://localhost:8529/_api/
   collection/UnitTestsCollectionBasics

HTTP/1.1 200 OK
content-type: application/json
```

```
{
  "name": "UnitTestsCollectionBasics",
  "code": 200,
  "id": 73482,
  "status": 3,
  "type": 2,
  "error": false
}
```

GET /_api/collection/{collection-name}/count,reads a collection with count

collection-name (string,required)

In addition to the above, the result also contains the number of documents. Note that this will always load the collection into memory.

- `count`: The number of documents inside the collection.

If the `collection-name` is missing, then a `HTTP 400` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

Using an identifier and requesting the number of documents:

```
> curl -X GET --dump - http://localhost:8529/_api/
   collection/73482/count

HTTP/1.1 200 OK
content-type: application/json

{
  "name": "UnitTestsCollectionBasics",
  "code": 200,
  "waitForSync": true,
  "isVolatile": false,
  "id": 73482,
  "journalSize": 33554432,
  "count": 0,
  "status": 3,
  "type": 2,
  "error": false
}
```

GET /_api/collection/{collection-name}/figures,reads a collection with stats

collection-name (string,required)

In addition to the above, the result also contains the number of documents and additional statistical information about the collection. Note that this will always load the collection into memory.

- `count`: The number of documents inside the collection.
- `figures.alive.count`: The number of living documents.
- `figures.alive.size`: The total size in bytes used by all living documents.
- `figures.dead.count`: The number of dead documents.
- `figures.dead.size`: The total size in bytes used by all dead documents.
- `figures.dead.deletion`: The total number of deletion markers.
- `figures.datafiles.count`: The number of active datafiles.
- `figures.datafiles.fileSize`: The total filesize of datafiles.
- `figures.journals.count`: The number of journal files.
- `figures.journals.fileSize`: The total filesize of journal files.
- `figures.shapes.count`: The total number of shapes used in the collection (this includes shapes that are not in use anymore)
- `figures.attributes.count`: The total number of attributes used in the collection (this includes attributes that are not in use anymore)
- `journalSize`: The maximal size of the journal in bytes.

Note: the filesizes of shapes and compactor files are not reported.

If the `collection-name` is missing, then a `HTTP 400` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

Using an identifier and requesting the figures of the collection:

```
> curl -X GET --dump - http://localhost:8529/_api/
   collection/73482/figures

HTTP/1.1 200 OK
content-type: application/json

{
  "name": "UnitTestsCollectionBasics",
  "code": 200,
  "figures": {
    "datafiles": {
      "count": 0,
      "fileSize": 0
    },
    "journals": {
      "count": 0,
      "fileSize": 33554432
    },
    "alive": {
      "size": 0,
      "count": 0
    },
    "dead": {
      "size": 2384,
      "count": 149
    },
    "shapes": {
      "count": 59
    },
  },
  "waitForSync": true,
  "isVolatile": false,
  "id": 73482,
  "journalSize": 134217728,
  "count": 0,
  "status": 3,
  "type": 2,
  "error": false
}
```

GET /_api/collection/{collection-name}/revision, reads a collection with revision id

collection-name (string,required)

In addition to the above, the result will also contain the collection's revision id. The revision id is a server-generated string that clients can use to check whether data in a collection has changed since the last revision check.

- `revision`: The collection revision id as a string.

If the `collection-name` is missing, then a `HTTP 400` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

GET /_api/collection,reads all collections

excludeSystem (boolean,optional)

Returns an object with an attribute `collections` containing a list of all collection descriptions. The same information is also available in the `names` as hash map with the collection names as keys.

By providing the optional URL parameter `excludeSystem` with a value of `true`, all system collections will be excluded from the response.

Return information about all collections:

```
> curl -X GET --dump - http://localhost:8529/_api/
   collection

HTTP/1.1 200 OK
content-type: application/json

{
  "collections": [
    {
      "name": "employees",
      "id": 64091684,
      "status": 3,
      "type": 2
    },
    {
      "name": "units",
      "id": 63043108,
      "status": 3,
      "type": 2
    }
  ],
  "code": 200,
  "names": {
    "units": {
      "name": "units",
      "id": 63043108,
      "status": 3,
      "type": 2
    },
    "employees": {
      "name": "employees",
      "id": 64091684,
      "status": 3,
      "type": 2
```

```
    }
  },
  "error": false
}
```

7.3.3 Modifying a Collection

PUT /_api/collection/{collection-name}/load,loads a collection

collection-name (string,required)

Loads a collection into memory. Returns the collection on success.

The request might optionally contain the following attribute:

- `count`: If set, this controls whether the return value should include the number of documents in the collection. Setting `count` to `false` may speed up loading a collection. The default value for `count` is `true`.

On success an object with the following attributes is returned:

- `id`: The identifier of the collection.
- `name`: The name of the collection.
- `count`: The number of documents inside the collection. This is only returned if the `count` input parameters is set to `true` or has not been specified.
- `status`: The status of the collection as number.
- `type`: The collection type. Valid types are:
  - 2: document collection
  - 3: edges collection

If the `collection-name` is missing, then a `HTTP 400` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

```
> curl -X PUT --dump - http://localhost:8529/_api/
   collection/55144253/load

HTTP/1.1 200 OK
content-type: application/json

{
  "name": "UnitTestsCollectionBasics",
  "code": 200,
  "id": 55144253,
```

```
  "count": 0,
  "status": 3,
  "type": 2,
  "error": false
}
```

PUT /_api/collection/{collection-name}/unload,unloads a collection

collection-name (string,required)

Removes a collection from memory. This call does not delete any documents. You can use the collection afterwards; in which case it will be loaded into memory, again. On success an object with the following attributes is returned:

- `id`: The identifier of the collection.
- `name`: The name of the collection.
- `status`: The status of the collection as number.
- `type`: The collection type. Valid types are:
  - 2: document collection
  - 3: edges collection

If the `collection-name` is missing, then a `HTTP 400` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

```
> curl -X PUT --dump - http://localhost:8529/_api/
   collection/57765693/unload

HTTP/1.1 200 OK
content-type: application/json

{
  "name": "UnitTestsCollectionBasics",
  "code": 200,
  "id": 57765693,
  "status": 4,
  "type": 2,
  "error": false
}
```

PUT /_api/collection/{collection-name}/properties,changes the properties of a collection

collection-name (string,required)

Changes the properties of a collection. Expects an object with the attribute(s)

- `waitForSync`: If `true` then creating or changing a document will wait until the data has been synchronised to disk.
- `journalSize`: Size (in bytes) for new journal files that are created for the collection.

If returns an object with the attributes

- `id`: The identifier of the collection.
- `name`: The name of the collection.
- `waitForSync`: The new value.
- `journalSize`: The new value.
- `status`: The status of the collection as number.
- `type`: The collection type. Valid types are:
  - 2: document collection
  - 3: edges collection

Note: some other collection properties, such as `type` or `isVolatile` cannot be changed once the collection is created.

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/collection/70109828/properties
{ "waitForSync" : true }

HTTP/1.1 200 OK
content-type: application/json

{
  "name": "UnitTestsCollectionBasics",
  "code": 200,
```

```
  "waitForSync": true,
  "isVolatile": false,
  "id": 70109828,
  "journalSize": 134217728,
  "status": 3,
  "type": 2,
  "error": false
}
```

PUT /_api/collection/{collection-name}/rename,renames a collection

collection-name (string,required)

Renames a collection. Expects an object with the attribute(s)

- `name`: The new name.

If returns an object with the attributes

- `id`: The identifier of the collection.
- `name`: The new name of the collection.
- `status`: The status of the collection as number.
- `type`: The collection type. Valid types are:
  - 2: document collection
  - 3: edges collection

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/collection/59230852/rename
{ "name" : "UnitTestsCollectionBasics2" }

HTTP/1.1 200 OK
content-type: application/json

{
  "name": "UnitTestsCollectionBasics2",
  "code": 200,
  "id": 59230852,
  "status": 3,
  "type": 2,
  "error": false
}
```

Chapter 8

HTTP Interface for Indexes

8.1 Indexes

This is an introduction to ArangoDB's Http interface for indexes in general. There are special sections for various index types.

Index: Indexes are used to allow fast access to documents. For each collection there is always the primary index which is a hash index for the document key. This index cannot be dropped or changed.

Edge collections will also have an automatically created edges index, which cannot be modified.

Most user-land indexes can be created by defining the names of the attributes which should be indexed. Some index types allow indexing just one attribute (e.g. fulltext index) whereas other index types allow indexing multiple attributes.

Indexing system attributes such as `_id`, `_key`, `_from`, and `_to` is not supported by any index type. Manually creating an index that relies on any of these attributes is unsupported.

Index Handle: An index handle uniquely identifies an index in the database. It is a string and consists of a collection name and an index identifier separated by `/`.

Geo Index: A geo index is used to find places on the surface of the earth fast.

Hash Index: A hash index is used to find documents based on examples.

Edges Index: An edges index is automatically created for edge collections. It contains connections between vertex documents and is invoked when the connecting edges of a vertex are queried. There is no way to explicitly create or delete edge indexes.

Skiplist Index: A skiplist is used to find ranges of documents.

Fulltext Index: A fulltext index can be used to find words, or prefixes of words inside documents. A fulltext index can be set on one attribute only, and will index all words contained in documents that have a textual value in this attribute. Only words with a (specifyable) minimum length are indexed. Word tokenisation is done using the word boundary analysis provided by libicu, which is taking into account the selected language provided at server start. Words are indexed in their lower-cased form. The index supports complete match queries (full words) and prefix queries. The basic operations (create, read, update, delete) for documents are mapped to the standard HTTP methods (`POST`, `GET`, `PUT`, `DELETE`).

8.2 Address of an Index

All indexes in ArangoDB have an unique handle. This index handle identifies an index and is managed by ArangoDB. All indexes are found under the URI

```
http://server:port/_api/index/index-handle
```

For example: Assume that the index handle is `demo/63563528` then the URL of that index is:

```
http://localhost:8529/_api/index/demo/63563528
```

8.3 Working with Indexes using HTTP

GET /_api/index,reads an index

GET /_api/index/`index-handle`

The result is an objects describing the index. It has at least the following attributes:

- `id`: The identifier of the index.
- `type`: The type of the collection.

All other attributes are type-dependent.

```
> curl -X GET --dump - http://localhost:8529/_api/
   index/117843216/0

HTTP/1.1 200 OK
content-type: application/json

{
```

```
  "code": 200,
  "fields": [
    "_id"
  ],
  "id": "117843216/0",
  "type": "primary",
  "error": false
}
```

POST /_api/index,creates an index

POST /_api/index?collection=`collection-name`

Creates a new index in the collection `collection-name`. Expects an object containing the index details.

The type of the index to be created must specified in the `type` attribute of the index details. Depending on the index type, additional other attributes may need to specified in the request in order to create the index.

See **Accessing Cap Constraints via Http** (p. 104), **Accessing Geo Indexes via Http** (p. 116), **Accessing Hash Indexes via Http** (p. 106), **Accessing Fulltext Indexes via Http** (p. 125), and **Accessing Skip-List Indexes via Http** (p. 112) for details.

Most indexes (a notable exception being the cap constraint) require the list of attributes to be indexed in the `fields` attribute of the index details. Depending on the index type, a single attribute or multiple attributes may be indexed.

Indexing system attributes such as `_id`, `_key`, `_from`, and `_to` is not supported by any index type. Manually creating an index that relies on any of these attributes is unsupported.

Some indexes can be created as unique or non-unique variants. Uniqueness can be controlled for most indexes by specifying the `unique` in the index details. Setting it to `true` will create a unique index. Setting it to `false` or omitting the `unique` attribute will create a non-unique index.

Note that the following index types do not support uniqueness, and using the `unique` attribute with these types may lead to an error:

- cap constraints
- fulltext indexes

If the index does not already exist and could be created, then a `HTTP 201` is returned. If the index already exists, then a `HTTP 200` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

Creating a unique constraint:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=102901008
{ "type" : "hash", "unique" : true, "fields" : [ "a",
   "b" ] }

HTTP/1.1 201 Created
content-type: application/json
```

```
{
  "code": 201,
  "fields": [
    "a",
    "b"
  ],
  "id": "102901008/168054969",
  "type": "hash",
  "isNewlyCreated": true,
  "unique": true,
  "error": false
}
```

Creating a hash index:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=105981200
{ "type" : "hash", "unique" : false, "fields" : [ "a",
    "b" ] }

HTTP/1.1 201 Created
content-type: application/json

{
  "code": 201,
  "fields": [
    "a",
    "b"
  ],
  "id": "105981200/171069625",
  "type": "hash",
  "isNewlyCreated": true,
  "unique": false,
  "error": false
}
```

Creating a skip-list:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=109061392
{ "type" : "skiplist", "unique" : false, "fields" : [
   "a", "b" ] }

HTTP/1.1 201 Created
content-type: application/json
```

```
{
  "code": 201,
  "fields": [
    "a",
    "b"
  ],
  "id": "109061392/173166777",
  "type": "skiplist",
  "isNewlyCreated": true,
  "unique": false,
  "error": false
}
```

Creating a unique skip-list:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=112141584
{ "type" : "skiplist", "unique" : true, "fields" : [ "
   a", "b" ] }

HTTP/1.1 201 Created
content-type: application/json

{
  "code": 201,
  "fields": [
    "a",
    "b"
  ],
  "id": "112141584/175722681",
  "type": "skiplist",
  "isNewlyCreated": true,
  "unique": true,
  "error": false
}
```

Creating a fulltext index:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=109061392
{ "type" : "fulltext", "fields" : [ "text" ] }

HTTP/1.1 201 Created
content-type: application/json
```

```
{
  "code": 201,
  "fields": [
    "text"
  ],
  "id": "109061392/1748352392",
  "type": "fulltext",
  "isNewlyCreated": true,
  "error": false
}
```

DELETE /_api/index,deletes an index

DELETE /_api/index/`index-handle`

Deletes an index with `index-handle`.

```
> curl -X DELETE --dump - http://localhost:8529/_api/
   index/180506809/181424313

HTTP/1.1 200 OK
content-type: application/json

{
  "code": 200,
  "id": "180506809/181424313",
  "error": false
}
```

GET /_api/index,reads all indexes of a collection

GET /_api/index?collection=`collection-name`

Returns an object with an attribute `indexes` containing a list of all index descriptions for the given collection. The same information is also available in the `identifiers` as hash map with the index handle as keys.

Return information about all indexes:

```
> curl -X GET --dump - http://localhost:8529/_api/
   index?collection=115221776

HTTP/1.1 200 OK
content-type: application/json

{
  "code": 200,
  "indexes": [
    {
      "fields": [
        "_id"
      ],
      "id": "115221776/0",
      "type": "primary"
    }
  ],
  "error": false,
  "identifiers": {
    "115221776/0": {
      "fields": [
        "_id"
      ],
      "id": "115221776/0",
      "type": "primary"
    }
  }
}
```

Chapter 9

Accessing Cap Constraints via Http

POST /_api/index,creates a cap constraint

POST /_api/index?collection=`collection-name`

Creates a cap constraint (**Introduction to Cap Constraints** (p. **??**)) for the collection `collection-name`, if it does not already exist. Expects an object containing the index details.

- `type`: must be equal to `"cap"`.
- `size`: The maximal number of documents for the collection.

Note that the cap constraint does not index particular attributes of the documents in a collection, but limits the number of documents in the collection to a maximum value. The cap constraint thus does not support attribute names specified in the `fields` attribute nor uniqueness of any kind via the `unique` attribute.

If the index does not already exist and could be created, then a `HTTP 201` is returned. If the index already exists, then a `HTTP 200` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

Creating a cap collection

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=1786279
{ "type" : "cap", "size" : 10 }

HTTP/1.1 201 Created
content-type: application/json

{
```

```
  "size": 10,
  "code": 201,
  "id": "1786279/2769319",
  "type": "cap",
  "isNewlyCreated": true,
  "error": false
}
```

Chapter 10

Accessing Hash Indexes via Http

If a suitable hash index exists, then **/_api/simple/by-example** (p. 54) will use this index to execute a query-by-example.

POST /_api/index,creates a hash index

POST /_api/index?collection=`collection-name`

Creates a hash index for the collection `collection-name`, if it does not already exist. The call expects an object containing the index details.

- `type`: must be equal to `"hash"`.
- `fields`: A list of attribute paths.
- `unique`: If `true`, then create a unique index.

If the index does not already exist and could be created, then a `HTTP 201` is returned. If the index already exists, then a `HTTP 200` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

If the collection already contains documents and you try to create a unique hash index in such a way that there are documents violating the uniqueness, then a `HTTP 400` is returned.

Creating an unique constraint:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=102901008
{ "type" : "hash", "unique" : true, "fields" : [ "a",
   "b" ] }

HTTP/1.1 201 Created
```

```
content-type: application/json

{
  "code": 201,
  "fields": [
    "a",
    "b"
  ],
  "id": "102901008/168054969",
  "type": "hash",
  "isNewlyCreated": true,
  "unique": true,
  "error": false
}
```

Creating a hash index:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=105981200
{ "type" : "hash", "unique" : false, "fields" : [ "a",
    "b" ] }

HTTP/1.1 201 Created
content-type: application/json

{
  "code": 201,
  "fields": [
    "a",
    "b"
  ],
  "id": "105981200/171069625",
  "type": "hash",
  "isNewlyCreated": true,
  "unique": false,
  "error": false
}
```

PUT /_api/simple/by-example,executes simple query "by-example"

PUT /_api/simple/by-example

This will find all documents matching a given example.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `example`: The example.
- `skip`: The documents to skip in the query. (optional)
- `limit`: The maximal amount of documents to return. (optional)

Returns a cursor containing the result, see **HTTP Interface for AQL Query Cursors** (p. 28) for details.

Matching an attribute:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/by-example
{ "collection" : "3179705695", "example" : [ { "i" : 1
    } ] }

HTTP/1.1 201 Created
content-type: application/json

{
  "result": [
    { "a": { "k": 2, "j": 2 }, "i": 1, "_rev":
       "3181802847", "_id": "3179705695/3181802847" },
    { "a": { "j": 1 }, "i": 1, "_rev": "3181475167", "
       _id": "3179705695/3181475167" },
    { "a": { "k": 1, "j": 1 }, "i": 1, "_rev":
       "3181737311", "_id": "3179705695/3181737311" },
    { "i": 1, "_rev": "3181147487", "_id":
       "3179705695/3181147487" }
  ],
  "count": 4,
  "error": false,
  "hasMore": false,
  "code": 201
}
```

Matching an attribute which is a sub-document:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/by-example
{ "collection" : "3179705695", "example" : [ { "a" : {
    "j" : 1 } } ] }

HTTP/1.1 201 Created
content-type: application/json

{
  "result": [
    { "a": { "j": 1 }, "i": 1, "_rev": "3181475167", "
       _id": "3179705695/3181475167" }
  ],
  "count": 1,
  "error": false,
  "hasMore": false,
  "code": 201
}
```

Matching an attribute within a sub-document:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/by-example
{ "collection" : "3179705695", "example" : [ "a.j", 1
   ] }

HTTP/1.1 201 Created
content-type: application/json

{
  "result": [
    { "a": { "j": 1 }, "i": 1, "_rev": "3181475167", "
       _id": "3179705695/3181475167" },
    { "a": { "k": 1, "j": 1 }, "i": 1, "_rev":
       "3181737311", "_id": "3179705695/3181737311" }
  ],
  "count": 2,
  "error": false,
  "hasMore": false,
  "code": 201
}
```

PUT /_api/simple/first-example,executes simple query "first-example"

PUT /_api/simple/first-example

This will return the first document matching a given example.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `example`: The example.

Returns a result containing the document or `HTTP 404` if no document matched the example.

If a matching document was found:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/first-example
{ "collection" : "666351134", "example" : [ "a.j", 1,
   "a.k", 1 ] }

HTTP/1.1 200 OK
content-type: application/json

{
  "error": false,
  "code": 200,
  "document": { "_rev": "668382750", "_id":
     "666351134/668382750", "a": { "k": 1, "j": 1, "l"
      : 10 }, "i": 1 }
}
```

If no document was found:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/first-example
{ "collection" : "666351134", "example" : [ "a.j", 1,
   "a.k", 2 ] }

HTTP/1.1 404 Not Found
content-type: application/json

{
  "errorMessage": "no match",
  "error": true,
  "code": 404,
```

```
  "errorNum": 404
}
```

Chapter 11

Accessing Skip-List Indexes via Http

If a suitable skip-list index exists, then **/_api/simple/range** (p. 59) will use this index to execute a range query.

POST /_api/index,creates a hash index

POST /_api/index?collection=`collection-name`

Creates a skip-list index for the collection `collection-name`, if it does not already exist. The call expects an object containing the index details.

- `type`: must be equal to `"skiplist"`.
- `fields`: A list of attribute paths.
- `unique`: If `true`, then create a unique index.

If the index does not already exist and could be created, then a `HTTP 201` is returned. If the index already exists, then a `HTTP 200` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

If the collection already contains documents and you try to create a unique skip-list index in such a way that there are documents violating the uniqueness, then a `HTTP 400` is returned.

Creating a skiplist:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=109061392
{ "type" : "skiplist", "unique" : false, "fields" : [
   "a", "b" ] }

HTTP/1.1 201 Created
```

```
content-type: application/json

{
  "code": 201,
  "fields": [
    "a",
    "b"
  ],
  "id": "109061392/173166777",
  "type": "skiplist",
  "isNewlyCreated": true,
  "unique": false,
  "error": false
}
```

PUT /_api/simple/range,executes simple range query

PUT /_api/simple/range

This will find all documents within a given range. You must declare a skip-list index on the attribute in order to be able to use a range query.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `attribute`: The attribute path to check.
- `left`: The lower bound.
- `right`: The upper bound.
- `closed`: If true, use interval including `left` and `right`, otherwise exclude `right`, but include `left`.
- `skip`: The documents to skip in the query. (optional)
- `limit`: The maximal amount of documents to return. (optional)

Returns a cursor containing the result, see **HTTP Interface for AQL Query Cursors** (p. 28) for details.

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/range
{ "collection" : "6328369086", "attribute" : "i", "
   left" : 2, "right" : 4 }

HTTP/1.1 201 Created
content-type: application/json

{
  "code": 201,
  "result": [
    { "_id": "6328369086/6329876414", "i": 2, "_rev":
       "6329876414" },
    { "_id": "6328369086/6329941950", "i": 3, "_rev":
       "6329941950" }
  ],
  "count": 2,
```

```
  "hasMore": false,
  "error": false
}
```

Chapter 12

Accessing Geo Indexes via Http

POST /_api/index,creates a geo-spatial index

POST /_api/index?collection=`collection-name`

Creates a geo-spatial index in the collection `collection-name`, if it does not already exist. Expects an object containing the index details.

- `type`: must be equal to `"geo"`.

- `fields`: A list with one or two attribute paths.

  If it is a list with one attribute path `location`, then a geo-spatial index on all documents is created using `location` as path to the coordinates. The value of the attribute must be a list with at least two double values. The list must contain the latitude (first value) and the longitude (second value). All documents, which do not have the attribute path or with value that are not suitable, are ignored.

  If it is a list with two attribute paths `latitude` and `longitude`, then a geo-spatial index on all documents is created using `latitude` and `longitude` as paths the latitude and the longitude. The value of the attribute `latitude` and of the attribute `longitude` must a double. All documents, which do not have the attribute paths or which values are not suitable, are ignored.

- `geoJson`: If a geo-spatial index on a `location` is constructed and `geoJson` is `true`, then the order within the list is longitude followed by latitude. This corresponds to the format described in `http://geojson.org/geojson-spec.html#positions`

- `constraint`: If `constraint` is `true`, then a geo-spatial constraint is created. The constraint is a non-unique variant of the index. Note that it

is also possible to set the `unique` attribute instead of the `constraint` attribute.

- `ignoreNull`: If a geo-spatial constraint is created and `ignoreNull` is true, then documents with a null in `location` or at least one null in `latitude` or `longitude` are ignored.

If the index does not already exist and could be created, then a `HTTP 201` is returned. If the index already exists, then a `HTTP 200` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

Creating a geo index with a location attribute:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=demo
{ "type" : "geo", "fields" : [ "b" ] }

HTTP/1.1 201 Created
content-type: application/json

{
  "code": 201,
  "geoJson": false,
  "fields": [
    "b"
  ],
  "id": "demo/162222265",
  "type": "geo1",
  "unique" : false,
  "constraint" : false,
  "isNewlyCreated": true,
  "error": false
}
```

Creating a geo index with latitude and longitude attributes:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=demo
{ "type" : "geo", "fields" : [ "e", "f" ] }

HTTP/1.1 201 Created
content-type: application/json

{
  "code": 201,
```

```
  "fields": [
    "e",
    "f"
  ],
  "id": "demo/165564601",
  "type": "geo2",
  "unique" : false,
  "constraint" : false,
  "isNewlyCreated": true,
  "error": false
}
```

PUT /_api/simple/near,executes simple query "near"

PUT /_api/simple/near

The default will find at most 100 documents near a given coordinate. The returned list is sorted according to the distance, with the nearest document coming first. If there are near documents of equal distance, documents are chosen randomly from this set until the limit is reached.

In order to use the `near` operator, a geo index must be defined for the collection. This index also defines which attribute holds the coordinates for the document. If you have more then one geo-spatial index, you can use the `geo` field to select a particular index.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `latitude`: The latitude of the coordinate.
- `longitude`: The longitude of the coordinate.
- `distance`: If given, the attribute key used to store the distance. (optional)
- `skip`: The number of documents to skip in the query. (optional)
- `limit`: The maximal amount of documents to return. The `skip` is applied before the `limit` restriction. The default is 100. (optional)
- `geo`: If given, the identifier of the geo-index to use. (optional)

Returns a cursor containing the result, see **HTTP Interface for AQL Query Cursors** (p. 28) for details.

Without distance:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/near
{ "collection" : "11172897562", "latitude" : 0, "
   longitude" : 0, "skip" : 1, "limit" : 2 }

HTTP/1.1 201 Created
content-type: application/json

{
```

```
  "hasMore": false,
  "count": 5,
  "result": [
    {
      "_id": "11172897562/11178271514",
      "loc": [ 0, -10 ],
      "_rev": "11178271514"
    },
    {
      "_id": "11172897562/11177616154",
      "loc": [ -10, 0 ],
      "_rev": "11177616154"
    }
  ],
  "code": 201,
  "error": false
}
```

With distance:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/near
{ "collection" : "11182465818", "latitude" : 0, "
   longitude" : 0, "limit" : 2, "distance" : "distance
   " }

HTTP/1.1 201 Created
content-type: application/json

{
  "hasMore": false,
  "count": 5,
  "result": [
    {
      "distance": 0,
      "_id": "11182465818/11187905306",
      "loc": [ 0, 0 ],
      "_rev": "11187905306"
    },
    {
      "distance": 1111949.26644559,
      "_id": "11182465818/11187839770",
      "loc": [ 0, -10 ],
      "_rev": "11187839770"
```

```
    }
  ],
  "code": 201,
  "error": false
}
```

PUT /_api/simple/within,executes simple query "within"

PUT /_api/simple/within

This will find all documents with in a given radius around the coordinate (`latitude`, `longitude`). The returned list is sorted by distance.

In order to use the `within` operator, a geo index must be defined for the collection. This index also defines which attribute holds the coordinates for the document. If you have more then one geo-spatial index, you can use the `geo` field to select a particular index.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `latitude`: The latitude of the coordinate.
- `longitude`: The longitude of the coordinate.
- `radius`: The maximal radius (in meters).
- `distance`: If given, the result attribute key used to store the distance values (optional). If specified, distances are returned in meters.
- `skip`: The documents to skip in the query. (optional)
- `limit`: The maximal amount of documents to return. (optional)
- `geo`: If given, the identifier of the geo-index to use. (optional)

Returns a cursor containing the result, see **HTTP Interface for AQL Query Cursors** (p. 28) for details.

Without distance:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/within
{ "collection" : "2076584128", "latitude" : 0, "
   longitude" : 0, "skip" : 1, "radius" : 1111950 }

HTTP/1.1 201 Created
content-type: application/json

{
```

```
  "result": [
    {
      "_id": "2076584128/2082744512",
      "loc": [ 10, 0 ],
      "_rev": "2082744512"
    },
    {
      "_id": "2076584128/2082089152",
      "loc": [ 0, 10 ],
      "_rev": "2082089152"
    },
    {
      "_id": "2076584128/2081302720",
      "loc": [ -10, 0 ],
      "_rev": "2081302720"
    },
    {
      "_id": "2076584128/2081958080",
      "loc": [ 0, -10 ],
      "_rev": "2081958080"
    }
  ],
  "code": 201,
  "hasMore": false,
  "count": 4,
  "error": false
}
```

With distance:

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/within
{ "collection" : "2086152384", "latitude" : 0, "
   longitude" : 0, "distance" : "distance", "radius" :
    1111950 }

HTTP/1.1 201 Created
content-type: application/json

{
  "result": [
    {
      "distance": 0,
      "_id": "2086152384/2091591872",
```

```
      "loc": [ 0, 0 ],
      "_rev": "2091591872"
    },
    {
      "distance": 1111949.26644559,
      "_id": "2086152384/2090870976",
      "loc": [ -10, 0 ],
      "_rev": "2090870976"
    },
    {
      "distance": 1111949.26644559,
      "_id": "2086152384/2091657408",
      "loc": [ 0, 10 ],
      "_rev": "2091657408"
    },
    {
      "distance": 1111949.26644559,
      "_id": "2086152384/2092312768",
      "loc": [ 10, 0 ],
      "_rev": "2092312768"
    },
    {
      "distance": 1111949.26644559,
      "_id": "2086152384/2091526336",
      "loc": [ 0, -10 ],
      "_rev": "2091526336"
    }
  ],
  "code": 201,
  "hasMore": false,
  "count": 5,
  "error": false
}
```

Chapter 13

Accessing Fulltext Indexes via Http

If a fulltext index exists, then **/_api/simple/fulltext** (p. 67) will use this index to execute the specified fulltext query.

POST /_api/index,creates a fulltext index

POST /_api/index?collection=`collection-name`

Creates a fulltext index for the collection `collection-name`, if it does not already exist. The call expects an object containing the index details.

- `type`: must be equal to `"fulltext"`.
- `fields`: A list of attribute names. Currently, the list is limited to exactly one attribute, so the value of `fields` should look like this for example: `[ "text" ]`.
- `minLength`: Minimum character length of words to index. Will default to a server-defined value if unspecified. It is thus recommended to set this value explicitly when creating the index.

If the index does not already exist and could be created, then a `HTTP 201` is returned. If the index already exists, then a `HTTP 200` is returned.

If the `collection-name` is unknown, then a `HTTP 404` is returned.

Creating a fulltext index:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/index?collection=109061392
{ "type" : "fulltext", "fields" : [ "text" ] }

HTTP/1.1 201 Created
```

```
content-type: application/json

{
  "code": 201,
  "fields": [
    "text"
  ],
  "id": "109061392/1748352392",
  "type": "fulltext",
  "isNewlyCreated": true,
  "error": false
}
```

PUT /_api/simple/fulltext,executes simple query "fulltext"

PUT /_api/simple/fulltext

This will find all documents from the collection that match the fulltext query specified in `query`.

In order to use the `fulltext` operator, a fulltext index must be defined for the collection and the specified attribute.

The call expects a JSON hash array as body with the following attributes:

- `collection`: The name of the collection to query.
- `attribute`: The attribute that contains the texts.
- `query`: The fulltext query.
- `skip`: The documents to skip in the query. (optional)
- `limit`: The maximal amount of documents to return. (optional)
- `index`: If given, the identifier of the fulltext-index to use. (optional)

Returns a cursor containing the result, see **HTTP Interface for AQL Query Cursors** (p. 28) for details.

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/simple/fulltext
{ "collection" : "2076584128", "attribute" : "text", "
   query" : "word" }

HTTP/1.1 201 Created
content-type: application/json

{
  "result": [
    {
      "_id": "2076584128/2082744512",
      "text" : "this text contains a word",
      "_rev": "2082744512"
    },
    {
      "_id": "2076584128/2082089152",
      "text" : "this text also contains a word",
```

```
      "loc": [ 0, 10 ],
      "_rev": "2082089152"
    }
  ],
  "code": 201,
  "hasMore": false,
  "count": 2,
  "error": false
}
```

Chapter 14

HTTP Interface for Transactions

14.1 Transactions

ArangoDB's transactions are executed on the server. Transactions can be initiated by clients by sending the transaction description for execution to the server.

Transactions in ArangoDB do not offer seperate `BEGIN`, `COMMIT` and `ROLLBACK` operations as they are available in many other database products. Instead, ArangoDB transactions are described by a Javascript function, and the code inside the Javascript function will then be executed transactionally. At the end of the function, the transaction is automatically committed, and all changes done by the transaction will be persisted. If an exception is thrown during transaction execution, all operations performed in the transaction are rolled back.

For a more detailed description of how transactions work in ArangoDB please refer to **Transactions** (p. **??**).

14.2 Executing Transactions via HTTP

POST /_api/transaction,executes a transaction

POST /_api/transaction

The transaction description must be passed in the body of the POST request.

The following attributes must be specified inside the JSON object:

- `collections`: contains the list of collections to be used in the transaction (mandatory). `collections` must be a JSON array that can have the optional sub-attributes `read` and `write`. `read` and `write` must each be either lists of collections names or strings with a single collection name.

- `action`: the actual transaction operations to be executed, in the form of stringified Javascript code. The code will be executed on server side, with late binding. It is thus critical that the code specified in `action` properly sets up all the variables it needs. If the code specified in `action` ends with a return statement, the value returned will also be returned by the REST API in the `result` attribute if the transaction committed successfully.

The following optional attributes may also be specified in the request:

- `waitForSync`: an optional boolean flag that, if set, will force the transaction to write all data to disk before returning.

- `lockTimeout`: an optional numeric value that can be used to set a timeout for waiting on collection locks. If not specified, a default value will be used. Setting `lockTimeout` to `0` will make ArangoDB not time out waiting for a lock.

- `params`: optional arguments passed to `action`.

If the transaction is fully executed and committed on the server, `HTTP 200` will be returned. Additionally, the return value of the code defined in `action` will be returned in the `result` attribute.

For successfully committed transactions, the returned JSON object has the following properties:

- `error`: boolean flag to indicate if an error occurred (`false` in this case)

- `code`: the HTTP status code

- `result`: the return value of the transaction

If the transaction specification is either missing or malformed, the server will respond with `HTTP 400`.

The body of the response will then contain a JSON object with additional error details. The object has the following attributes:

- `error`: boolean flag to indicate that an error occurred (`true` in this case)

- `code`: the HTTP status code

- `errorNum`: the server error number

- `errorMessage`: a descriptive error message

If a transaction fails to commit, either by an exception thrown in the `action` code, or by an internal error, the server will respond with an error.

Exceptions thrown by users will make the server respond with a return code of `HTTP 500`. Any other errors will be returned with any of the return codes `HTTP 400`, `HTTP 409`, or `HTTP 500`.

Executing a transaction on a single collection:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/transaction
{ "collections" : { "write" : "users" }, "action" : "
   function () { var users = require(\"internal\").db.
   users; users.save({ _key: \"hello\" }); return
   users.count(); }" }

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8

{ "result" : 1, "error" : false, "code" : 200 }
```

Executing a transaction using multiple collections:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/transaction
{ "collections" : { "write" : [ "users", "logins" ] },
    "action" : "function () { var users = require(\"
   internal\").db.users; var logins = require(\"
   internal\").db.logins; users.save({ }); logins.save
   ({ }); return \"worked!\"; }" }

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8

{ "result" : "worked!", "error" : false, "code" : 200
   }
```

Aborting a transaction due to an internal error:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/transaction
{ "collections" : { "write" : [ "users" ] }, "action"
   : "function () { var users = require(\"internal\").
   db.users; users.save({ _key: \"abc\" }); users.save
   ({ _key: \"abc\" }); }" }
```

```
HTTP/1.1 400 Bad Request
content-type: application/json; charset=utf-8

{ "error" : true, "code" : 400, "errorNum" : 1210, "
   errorMessage" : "cannot save document: unique
   constraint violated" }
```

Aborting a transaction by throwing an exception:

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/transaction
{ "collections" : { "read" : "users" }, "action" : "
   function () { throw \"doh!\"; }" }

HTTP/1.1 500 Internal Error
content-type: application/json; charset=utf-8

{ "error" : true, "code" : 500, "errorNum" : 500, "
   errorMessage" : "doh!" }
```

Chapter 15

HTTP Interface for Graphs

POST /_api/graph,create graph

waitForSync (boolean,optional) Wait until document has been sync to disk.

Creates a new graph.

The call expects a JSON hash array as body with the following attributes:

- `_key`: The name of the new graph.
- `vertices`: The name of the vertices collection.
- `edges`: The name of the egde collection.

Returns an object with an attribute `graph` containing a list of all graph properties.

is returned if the graph was created sucessfully and `waitForSync` was `true`.

is returned if the graph was created sucessfully and `waitForSync` was `false`.

is returned if it failed. The response body contains an error document in this case.

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/graph
{"_key" : "graph1", "vertices" : "v", "edges" : "e"}

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
etag: 11767958

{
  "graph": {
    "_id": "_graphs/graph1",
    "_rev": "11767958",
```

```
    "_key": "graph1",
    "vertices": "v",
    "edges": "e"
  },
  "error": false,
  "code": 201
}
```

GET /_api/graph,get graph properties

rev (string,optional) Revision of a graph

Returns an object with an attribute `graph` containing a list of all graph properties.

If the "If-None-Match" header is given, then it must contain exactly one etag. The document is returned, if it has a different revision than the given etag. Otherwise a `HTTP 304` is returned.

If the "If-Match" header is given, then it must contain exactly one etag. The document is returned, if it has the same revision ad the given etag. Otherwise a `HTTP 412` is returned. As an alternative you can supply the etag in an attribute `rev` in the URL.

is returned if the graph was found

is returned if the graph was not found. The response body contains an error document in this case.

"If-None-Match" header is given and the current graph has not a different version

"If-Match" header or `rev` is given and the current graph has a different version

get graph by name

```
> curl -X GET --dump - http://localhost:8529/_api/
   graph/graph1

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: 11767958

{
  "graph": {
    "_id": "_graphs/graph1",
    "_rev": "11767958",
    "_key": "graph1",
    "vertices": "v",
    "edges": "e"
  },
  "error": false,
  "code": 200
}
```

DELETE /_api/graph,delete graph

waitForSync (boolean,optional) Wait until document has been sync to disk.

rev (string,optional) Revision of a graph

Deletes graph, edges and vertices

If the "If-Match" header is given, then it must contain exactly one etag. The document is deleted, if it has the same revision ad the given etag. Otherwise a `HTTP 412` is returned. As an alternative you can supply the etag in an attribute `rev` in the URL.

is returned if the graph was deleted

is returned if the graph was not found. The response body contains an error document in this case.

"If-Match" header or `rev` is given and the current graph has a different version

```
> curl -X DELETE --dump - http://localhost:8529/_api/
   graph/graph1

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8

{
  "deleted": true,
  "error": false,
  "code": 200
}
```

POST /_api/graph/graph-name/vertex,create vertex

waitForSync (boolean,optional) Wait until document has been sync to disk.

Creates a vertex in a graph.

The call expects a JSON hash array as body with the vertex properties:

- _key: The name of the vertex (optional).
- further optional attributes.

Returns an object with an attribute vertex containing a list of all vertex properties.

is returned if the graph was created sucessfully and waitForSync was true.

is returned if the graph was created sucessfully and waitForSync was false.

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/graph/graph1/vertex
{"_key" : "v1", "optional1" : "val1", "optional2" : "
   val2"}

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
etag: 24332177

{
  "vertex": {
    "_id": "v/v1",
    "_rev": "24332177",
    "_key": "v1",
    "optional1": "val1",
    "optional2": "val2"
  },
  "error": false,
  "code": 201
}
```

GET /_api/graph/graph-name/vertex,get vertex

rev (string,optional) Revision of a vertex

Returns an object with an attribute vertex containing a list of all vertex properties.

If the "If-None-Match" header is given, then it must contain exactly one etag. The document is returned, if it has a different revision than the given etag. Otherwise a HTTP 304 is returned.

If the "If-Match" header is given, then it must contain exactly one etag. The document is returned, if it has the same revision ad the given etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the etag in an attribute rev in the URL.

is returned if the graph was found

"If-Match" header is given and the current graph has not a different version

is returned if the graph or vertex was not found. The response body contains an error document in this case.

"If-None-Match" header or rev is given and the current graph has a different version

get vertex properties by name

```
> curl -X GET --dump - http://localhost:8529/_api/
   graph/graph1/vertex/v1

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: 24332177

{
  "vertex": {
    "_id": "v/v1",
    "_rev": "24332177",
    "_key": "v1",
    "optional1": "val1",
    "optional2": "val2"
  },
  "error": false,
  "code": 200
}
```

PUT /_api/graph/graph-name/vertex,update vertex

waitForSync (boolean,optional) Wait until vertex has been sync to disk.

rev (string,optional) Revision of a vertex

Replaces the vertex properties.

The call expects a JSON hash array as body with the new vertex properties.

Returns an object with an attribute vertex containing a list of all vertex properties.

If the "If-Match" header is given, then it must contain exactly one etag. The document is updated, if it has the same revision ad the given etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the etag in an attribute rev in the URL.

is returned if the vertex was updated sucessfully and waitForSync was true.

is returned if the vertex was updated sucessfully and waitForSync was false.

is returned if the graph or the vertex was not found. The response body contains an error document in this case.

"If-Match" header or rev is given and the current vertex has a different version

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/graph/graph1/vertex/v1
{"optional1" : "val2"}

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: 24332190

{
  "vertex": {
    "_id": "v/v1",
    "_rev": "24332190",
    "_key": "v1",
    "optional1": "val2"
  },
  "error": false,
  "code": 200
}
```

PATCH /_api/graph/graph-name/vertex,update vertex

waitForSync (boolean,optional) Wait until vertex has been sync to disk.

rev (string,optional) Revision of a vertex

keepNull (boolean,optional) Modify the behavior of the patch command to remove any attribute

Partially updates the vertex properties.

The call expects a JSON hash array as body with the properties to patch.

Setting an attribute value to `null` in the patch document will cause a value of `null` be saved for the attribute by default. If the intention is to delete existing attributes with the patch command, the URL parameter `keepNull` can be used with a value of `false`. This will modify the behavior of the patch command to remove any attributes from the existing document that are contained in the patch document with an attribute value of `null`. If the "If-Match" header is given, then it must contain exactly one etag. The document is updated, if it has the same revision ad the given etag. Otherwise a `HTTP 412` is returned. As an alternative you can supply the etag in an attribute `rev` in the URL.

Returns an object with an attribute `vertex` containing a list of all vertex properties.

is returned if the vertex was updated sucessfully and `waitForSync` was `true`.

is returned if the vertex was updated sucessfully and `waitForSync` was `false`.

is returned if the graph or the vertex was not found. The response body contains an error document in this case.

"If-Match" header or `rev` is given and the current vertex has a different version

```
> curl --data @- -X PATCH --dump - http://localhost
   :8529/_api/graph/graph1/vertex/v1
{"optional2" : "vertexPatch2"}

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: 24332193

{
  "vertex": {
    "_id": "v/v1",
    "_rev": "24332193",
    "_key": "v1",
    "optional1": "val2",
    "optional2": "vertexPatch2"
  },
```

```
  "error": false,
  "code": 200
}

> curl --data @- -X PATCH --dump - http://localhost
   :8529/_api/graph/graph1/vertex/v1?keepNull=false
{"optional2" : null}

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: 24332199

{
  "vertex": {
    "_id": "v/v1",
    "_rev": "24332199",
    "_key": "v1",
    "optional1": "val2"
  },
  "error": false,
  "code": 200
}
```

DELETE /_api/graph/`graph-name`/vertex,delete vertex

waitForSync (boolean,optional) Wait until document has been sync to disk.

rev (string,optional) Revision of a vertex

Deletes vertex and all in and out edges of the vertex

If the "If-Match" header is given, then it must contain exactly one etag. The document is deleted, if it has the same revision ad the given etag. Otherwise a `HTTP 412` is returned. As an alternative you can supply the etag in an attribute `rev` in the URL.

is returned if the vertex was deleted

is returned if the graph or the vertex was not found. The response body contains an error document in this case.

"If-Match" header or `rev` is given and the current vertex has a different version

```
> curl -X DELETE --dump - http://localhost:8529/_api/
   graph/graph1/vertex/v1

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8

{
  "deleted": true,
  "error": false,
  "code": 200
}
```

POST /_api/graph/graph-name/edge,create edge

waitForSync (boolean,optional) Wait until edge has been sync to disk.

Creates an edge in a graph.

The call expects a JSON hash array as body with the edge properties:

- _key: The name of the edge.
- _from: The name of the from vertex.
- _to: The name of the to vertex.
- $label: A label for the edge (optional).
- further optional attributes.

Returns an object with an attribute edge containing the list of all edge properties.

is returned if the edge was created sucessfully and waitForSync was true.

is returned if the edge was created sucessfully and waitForSync was false.

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/graph/graph1/edge
{"_key" : "edge1", "_from" : "vert2", "_to" : "vert1",
    "optional1" : "val1"}

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
etag: 57595366

{
  "edge": {
    "_id": "e/edge1",
    "_rev": "57595366",
    "_key": "edge1",
    "_from": "v/vert2",
    "_to": "v/vert1",
    "$label": null,
    "optional1": "val1"
  },
  "error": false,
  "code": 201
}
```

GET /_api/graph/graph-name/edge,get edge

rev (string,optional) Revision of an edge

Returns an object with an attribute edge containing a list of all edge properties.

If the "If-None-Match" header is given, then it must contain exactly one etag. The document is returned, if it has a different revision than the given etag. Otherwise a HTTP 304 is returned.

If the "If-Match" header is given, then it must contain exactly one etag. The document is returned, if it has the same revision ad the given etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the etag in an attribute rev in the URL.

is returned if the edge was found

"If-Match" header is given and the current edge has not a different version

is returned if the graph or edge was not found. The response body contains an error document in this case.

"If-None-Match" header or rev is given and the current edge has a different version

```
> curl -X GET --dump - http://localhost:8529/_api/
   graph/graph1/edge/edge1

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: 57595366

{
  "edge": {
    "_id": "e/edge1",
    "_rev": "57595366",
    "_key": "edge1",
    "_from": "v/vert2",
    "_to": "v/vert1",
    "$label": null,
    "optional1": "val1"
  },
  "error": false,
  "code": 200
}
```

PUT /_api/graph/graph-name/edge,update edge

waitForSync (boolean,optional) Wait until edge has been sync to disk.

rev (string,optional) Revision of an edge

Replaces the optional edge properties.

The call expects a JSON hash array as body with the new edge properties.

If the "If-Match" header is given, then it must contain exactly one etag. The document is returned, if it has the same revision ad the given etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the etag in an attribute rev in the URL.

Returns an object with an attribute edge containing a list of all edge properties.

is returned if the edge was updated sucessfully and waitForSync was true.

is returned if the edge was updated sucessfully and waitForSync was false.

is returned if the graph or the edge was not found. The response body contains an error document in this case.

"If-Match" header or rev is given and the current edge has a different version

```
> curl --data @- -X PUT --dump - http://localhost
   :8529/_api/graph/graph1/edge/edge1
{"optional2" : "val2"}

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: 57595391

{
  "edge": {
    "_id": "e/edge1",
    "_rev": "57595391",
    "_key": "edge1",
    "_from": "v/vert2",
    "_to": "v/vert1",
    "$label": null,
    "optional2": "val2"
  },
  "error": false,
  "code": 200
}
```

PATCH /_api/graph/graph-name/edge,update edge

waitForSync (boolean,optional) Wait until edge has been sync to disk.

rev (string,optional) Revision of an edge

keepNull (boolean,optional) Modify the behavior of the patch command to remove any attribute

Partially updates the edge properties.

The call expects a JSON hash array as body with the properties to patch.

Setting an attribute value to `null` in the patch document will cause a value of `null` be saved for the attribute by default. If the intention is to delete existing attributes with the patch command, the URL parameter `keepNull` can be used with a value of `false`. This will modify the behavior of the patch command to remove any attributes from the existing document that are contained in the patch document with an attribute value of `null`.

If the "If-Match" header is given, then it must contain exactly one etag. The document is returned, if it has the same revision ad the given etag. Otherwise a `HTTP 412` is returned. As an alternative you can supply the etag in an attribute `rev` in the URL. Returns an object with an attribute `edge` containing a list of all edge properties.

is returned if the edge was updated sucessfully and `waitForSync` was `true`.

is returned if the edge was updated sucessfully and `waitForSync` was `false`.

is returned if the graph or the edge was not found. The response body contains an error document in this case.

"If-Match" header or `rev` is given and the current edge has a different version

```
> curl --data @- -X PATCH --dump - http://localhost
   :8529/_api/graph/graph1/edge/edge1
{"optional3" : "val3"}

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
etag: 57595398

{
  "edge": {
    "_id": "e/edge1",
    "_rev": "57595398",
    "_key": "edge1",
    "_from": "v/vert2",
    "_to": "v/vert1",
    "$label": null,
```

```
    "optional2": "val2",
    "optional3": "val3"
  },
  "error": false,
  "code": 200
}
```

DELETE /_api/graph/graph-name/edge,delete edge

waitForSync (boolean,optional) Wait until edge has been sync to disk.

rev (string,optional) Revision of an edge

Deletes an edge of the graph

If the "If-Match" header is given, then it must contain exactly one etag. The document is returned, if it has the same revision ad the given etag. Otherwise a HTTP 412 is returned. As an alternative you can supply the etag in an attribute rev in the URL.

is returned if the edge was deletd sucessfully and waitForSync was true.

is returned if the edge was deleted sucessfully and waitForSync was false.

is returned if the graph or the edge was not found. The response body contains an error document in this case.

"If-Match" header or rev is given and the current edge has a different version

```
> curl -X DELETE --dump - http://localhost:8529/_api/
   graph/graph1/edge/edge1

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8

{
  "deleted": true,
  "error": false,
  "code": 200
}
```

POST /_api/graph/graph-name/vertices/vertice-name,get vertices

Returns a cursor.

The call expects a JSON hash array as body to filter the result:

- batchSize: the batch size of the returned cursor
- limit: limit the result size
- count: return the total number of results (default "false")
- filter: a optional filter

The attributes of filter

- direction: Filter for inbound (value "in") or outbound (value "out") neighbors. Default value is "any".
- labels: filter by an array of edge labels (empty array means no restriction)
- properties: filter neighbors by an array of edge properties

The attributes of a property filter

- key: filter the result vertices by a key value pair
- value: the value of the key
- compare: a compare operator

is returned if the cursor was created

Select all vertices

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/grap/graph1/vertices/id2
{"batchSize" : 100, "filter" : { "direction" : "any"
   }}

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8

{
  "result": [
    {
      "_id": "v/id3",
      "_rev": "47810529",
```

```
      "_key": "id3",
      "optional2": 2,
      "optional1": "val1"
    },
    {
      "_id": "v/id1",
      "_rev": "47548385",
      "_key": "id1",
      "optional2": 1,
      "optional1": "val1"
    }
  ],
  "hasMore": false,
  "error": false,
  "code": 201
}
```

Select vertices by direction and property filter

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/grap/graph1/vertices/id2
{
  "batchSize" : 100,
  limit : 10,
  count : true,
  "filter" : {
    "direction" : "out",
    "properties" : [
      {
        "key" : "optional1",
        "value" : "val1",
        "compare" : "=="
      },
      {
        "key" : "optional2",
        "value" : 2,
        "compare" : "=="
      }
    ]
  }
}

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8
```

```
{
  "result": [
    {
      "_id": "v/id3",
      "_rev": "47810529",
      "_key": "id3",
      "optional2": 2,
      "optional1": "val1"
    }
  ],
  "hasMore": false,
  "error": false,
  "count" : 1,
  "code": 201
}
```

POST /_api/graph/graph-name/edges/vertex-name,get edges

POST /_api/graph/graph-name/edges/vertex-name

Returns a cursor.

The call expects a JSON hash array as body to filter the result:

- batchSize: the batch size of the returned cursor
- limit: limit the result size
- count: return the total number of results (default "false")
- filter: a optional filter

The attributes of filter

- direction: Filter for inbound (value "in") or outbound (value "out") neighbors. Default value is "any".
- labels: filter by an array of edge labels
- properties: filter neighbors by an array of properties

The attributes of a property filter

- key: filter the result vertices by a key value pair
- value: the value of the key
- compare: a compare operator

is returned if the cursor was created

Select all edges

```
> curl --data @- -X POST --dump - http://localhost
   :8529/_api/graph/graph1/edges/id2
{"batchSize" : 100, "filter" : { "direction" : "any"
   }}

HTTP/1.1 201 Created
content-type: application/json; charset=utf-8

{
  "result": [
    {
```

```
      "_id": "e/edge1",
      "_rev": "125407382",
      "_key": "edge1",
      "_from": "v/id1",
      "_to": "v/id2",
      "$label": null,
      "optional1": "val1a"
    },
    {
      "_id": "e/edge2",
      "_rev": "125800598",
      "_key": "edge2",
      "_from": "v/id2",
      "_to": "v/id3",
      "$label": null,
      "optional1": "val1b"
    }
  ],
  "hasMore": false,
  "error": false,
  "code": 201
}
```

Chapter 16

HTTP Interface for bulk imports

ArangoDB provides an HTTP interface to import multiple documents at once into a collection. This is known as a bulk import.

The data uploaded must be provided in JSON format. There are two mechanisms to import the data:

- self-contained documents: in this case, each document contains all attribute names and values. Attribute names may be completely different among the documents uploaded
- attribute names plus document data: in this case, the first document must be a JSON list containing the attribute names of the documents that follow. The following documents must be lists containing only the document data. Data will be mapped to the attribute names by attribute positions.

The endpoint address is `/_api/import` for both input mechanisms. Data must be sent to this URL using an HTTP POST request. The data to import must be contained in the body of the POST request.

The `collection` URL parameter must be used to specify the target collection for the import. The optional URL parameter `createCollection` can be used to create a non-existing collection during the import. If not used, importing data into a non-existing collection will produce an error. Please note that the `createCollection` flag can only be used to create document collections, not edge collections.

16.1 Importing self-contained documents

This import method allows uploading self-contained JSON documents. The documents must be uploaded in the body of the HTTP POST request. Each line of

the body will be interpreted as one stand-alone document. Empty lines in the body are allowed but will be skipped. Using this format, the documents are imported line-wise.

Example input data: { "_key": "key1", ... } { "_key": "key2", ... } ...

To use this method, the `type` URL parameter should be set to `documents`.

It is also possible to upload self-contained JSON documents that are embedded into a JSON list. Each element from the list will be treated as a document and be imported.

Example input data for this case:

```
[
  { "_key": "key1", ... },
  { "_key": "key2", ... },
  ...
]
```

This format does not require each document to be on a separate line, and any whitespace in the JSON data is allowed. It can be used to import a JSON-formatted result list (e.g. from arangosh) back into ArangoDB. Using this format requires ArangoDB to parse the complete list and keep it in memory for the duration of the import. This might be more resource-intensive than the line-wise processing.

To use this method, the `type` URL parameter should be set to `array`.

```
curl --data-binary @- -X POST --dump - "http://
   localhost:8529/_api/import?type=documents&
   collection=test&createCollection=true"
{ "name" : "test", "gender" : "male", "age" : 39 }
{ "type" : "bird", "name" : "robin" }

HTTP/1.1 201 Created
server: triagens GmbH High-Performance HTTP Server
connection: Keep-Alive
content-type: application/json; charset=utf-8

{"error":false,"created":2,"errors":0}
```

The server will respond with an HTTP 201 if everything went well. The number of documents imported will be returned in the `created` attribute of the response. If any documents were skipped or incorrectly formatted, this will be returned in the `errors` attribute.

16.2 Importing headers and values

When using this type of import, the attribute names of the documents to be imported are specified separate from the actual document value data. The first line of the HTTP POST request body must be a JSON list containing the attribute names for the documents that follow. The following lines are interpreted as the document data. Each document must be a JSON list of values. No attribute names are needed or allowed in this data section.

```
curl --data-binary @- -X POST --dump - "http://
   localhost:8529/_api/import?collection=test&
   createCollection=true"
[ "firstName", "lastName", "age", "gender" ]
[ "Joe", "Public", 42, "male" ]
[ "Jane", "Doe", 31, "female" ]

HTTP/1.1 201 Created
server: triagens GmbH High-Performance HTTP Server
connection: Keep-Alive
content-type: application/json; charset=utf-8

{"error":false,"created":2,"errors":0}
```

The server will again respond with an HTTP 201 if everything went well. The number of documents imported will be returned in the `created` attribute of the response. If any documents were skipped or incorrectly formatted, this will be returned in the `errors` attribute.

16.3 Importing in edge collections

Please note that when importing documents into an edge collection, it is mandatory that all imported documents contain the `_from` and `_to` attributes, and that these contain valid references.

Please also note that it is not possible to create a new edge collection on the fly using the `createCollection` parameter.

Chapter 17

HTTP Interface for Batch Requests

Clients normally send individual operations to ArangoDB in individual HTTP requests. This is straightforward and simple, but has the disadvantage that the network overhead can be significant if many small requests are issued in a row.

To mititage this problem, ArangoDB offers a batch request API that clients can use to send multiple operations in one batch to ArangoDB. This method is especially useful when the client has to send many HTTP requests with a small body/payload and the individual request resulsts do not depend on each other.

Clients can use ArangoDB's batch API by issuing a multipart HTTP POST request to the URL `/_api/batch` handler. The handler will accept the request if the Content-Type is `multipart/form-data` and a boundary string is specified. ArangoDB will then decompose the batch request into its individual parts using this boundary. This also means that the boundary string itself must not be contained in any of the parts. When ArangoDB has split the multipart request into its individual parts, it will process all parts sequentially as if it were a standalone request. When all parts are processed, ArangoDB will generate a multipart HTTP response that contains one part for each part operation result. For example, if you send a multipart request with 5 parts, ArangoDB will send back a multipart response with 5 parts as well.

The server expects each part message to start with exactly the following "header":

```
Content-Type: application/x-arango-batchpart
```

You can optionally specify a `Content-Id` "header" to uniquely identify each part message. The server will return the `Content-Id` in its response if it is specified. Otherwise, the server will not send a Content-Id "header" back. The server will not validate the uniqueness of the Content-Id. After the mandatory `Content-Type` and the optional `Content-Id` header, two Windows linebreaks (i.e. `\\r\\n\\r\\n`) must follow. Any deviation of this structure might lead to the part being rejected or incorrectly interpreted. The part request payload, formatted as a regular HTTP

request, must follow the two Windows linebreaks literal directly.

Note that the literal `Content-Type: application/x-arango-batchpart` technically is the header of the MIME part, and the HTTP request (including its headers) is the body part of the MIME part.

An actual part request should start with the HTTP method, the called URL, and the HTTP protocol version as usual, followed by arbitrary HTTP headers. Its body should follow after the usual `\\r\\n\\r\\n` literal. Part requests are therefore regular HTTP requests, only embedded inside a multipart message.

The following example will send a batch with 3 individual document creation operations. The boundary used in this example is `XXXsubpartXXX`.

```
> curl -X POST --data-binary @- --header "Content-Type
   : multipart/form-data; boundary=XXXsubpartXXX" http
   ://localhost:8529/_api/batch
--XXXsubpartXXX
Content-Type: application/x-arango-batchpart
Content-Id: 1

POST /_api/document?collection=xyz&createCollection=
   true HTTP/1.1

{"a":1,"b":2,"c":3}
--XXXsubpartXXX
Content-Type: application/x-arango-batchpart
Content-Id: 2

POST /_api/document?collection=xyz HTTP/1.1

{"a":1,"b":2,"c":3,"d":4}
--XXXsubpartXXX
Content-Type: application/x-arango-batchpart
Content-Id: 3

POST /_api/document?collection=xyz HTTP/1.1

{"a":1,"b":2,"c":3,"d":4,"e":5}
--XXXsubpartXXX--
```

The server will then respond with one multipart message, containing the overall status and the individual results for the part operations. The overall status should be 200 except there was an error while inspecting and processing the multipart message. The overall status therefore does not indicate the success of each part operation, but only indicates whether the multipart message could be handled suc-

cessfully.

Each part operation will return its own status value. As the part operation results are regular HTTP responses (just included in one multipart response), the part operation status is returned as a HTTP status code. The status codes of the part operations are exactly the same as if you called the individual operations standalone. Each part operation might also return arbitrary HTTP headers and a body/payload:

```
HTTP/1.1 200 OK
connection: Keep-Alive
content-type: multipart/form-data; boundary=
   XXXsubpartXXX
content-length: 1055

--XXXsubpartXXX
Content-Type: application/x-arango-batchpart
Content-Id: 1

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
etag: "9514299"
content-length: 53

{"error":false,"_id":"101059/9514299","_rev
   ":"9514299"}
--XXXsubpartXXX
Content-Type: application/x-arango-batchpart
Content-Id: 2

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
etag: "9579835"
content-length: 53

{"error":false,"_id":"101059/9579835","_rev
   ":"9579835"}
--XXXsubpartXXX
Content-Type: application/x-arango-batchpart
Content-Id: 3

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
etag: "9645371"
content-length: 53
```

```
{"error":false,"_id":"101059/9645371","_rev
   ":"9645371"}
--XXXsubpartXXX--
```

In the above example, the server returned an overall status code of 200, and each part response contains its own status value (202 in the example):

When constructing the multipart HTTP response, the server will use the same boundary that the client supplied. If any of the part responses has a status code of 400 or greater, the server will also return an HTTP header `x-arango-errors` containing the overall number of part requests that produced errors:

```
> curl -X POST --data-binary @- --header "Content-Type
   : multipart/form-data; boundary=XXXsubpartXXX" http
   ://localhost:8529/_api/batch
--XXXsubpartXXX
Content-Type: application/x-arango-batchpart

POST /_api/document?collection=nonexisting

{"a":1,"b":2,"c":3}
--XXXsubpartXXX
Content-Type: application/x-arango-batchpart

POST /_api/document?collection=xyz

{"a":1,"b":2,"c":3,"d":4}
--XXXsubpartXXX--
```

In this example, the overall response code is 200, but as some of the part request failed (with status code 404), the `x-arango-errors` header of the overall response is `1`:

```
HTTP/1.1 200 OK
x-arango-errors: 1
content-type: multipart/form-data; boundary=
   XXXsubpartXXX
content-length: 711

--XXXsubpartXXX
Content-Type: application/x-arango-batchpart

HTTP/1.1 404 Not Found
content-type: application/json; charset=utf-8
content-length: 111
```

```
{"error":true,"code":404,"errorNum":1203,"errorMessage
   ":"collection \/_api\/collection\/nonexisting not
   found"}
--XXXsubpartXXX
Content-Type: application/x-arango-batchpart

HTTP/1.1 202 Accepted
content-type: application/json; charset=utf-8
etag: "9841979"
content-length: 53

{"error":false,"_id":"101059/9841979","_rev
   ":"9841979"}
--XXXsubpartXXX--
```

Chapter 18

HTTP Interface for Administration and Monitoring

This is an introduction to ArangoDB's Http interface for administration and monitoring of the server.

GET /_admin/log,reads the log information

GET /_admin/log

Returns all fatal, error, warning or info log messages.

The returned object contains the attributes:

- `lid`: a list of log-entry identifiers. Each log message is uniquely identified by its `lid` and the identifiers are in ascending order.
- `level`: a list of the log-level of the log-entry.
- `timestamp`: a list of the timestamp as seconds since 1970-01-01 of the log-entry.
- `text`: a list of the text of the log-entry.
- `totalAmount`: the total amount of log entries before pagination.

GET /_admin/log?upto=log-level

Returns all log entries upto log-level. Note that log-level must be:

- `fatal` / `0`

- `error / 1`
- `warning / 2`
- `info / 3`
- `debug / 4`

GET /_admin/log?level=log-level

Returns all log entries of log-level. Note that `level=` and `upto=` are mutably exclusive.

GET /_admin/log?size=size&offset= offset

Paginates the result. Skip the first offset entries and limit the number of returned log-entries to size.

GET /_admin/log?start=lid

Returns all log entries such that their log-entry identifier is greater or equal to `lid`.

GET /_admin/log?sort=direction

Sort the log-entries either ascending if direction is `asc`, or descending if it is `desc` according to their `lid`. Note that the `lid` imposes a chronological order.

GET /_admin/log?search=text

Only return the log-entries containing the text string.

POST /_admin/modules/flush,flushs the module cache

POST /_admin/modules/flush

The call flushes the modules cache on the server. See **Modules Cache** (p. **??**) for details about this cache.

POST /_admin/routing/reload,reloads the routing collection

POST /_admin/routing/reload

Reloads the routing information from the collection `routing`.

GET /_admin/statistics,reads the statistics

Returns the statistics information. The returned objects contains the statistics figures group together according to the description returned by _admin/statistics-description. For instance, to access a figure userTime from the group system, you first select the sub-object describing the group stored in system and in that sub-object the value for userTime is stored in the attribute of the same name.

In case of a distribution, the returned object contains the total count in count and the distribution list in counts. The sum (or total) of the individual values is returned in sum.

Statistics were returned successfully.

```
unix> curl --dump - http://localhost:8529/_admin/
   statistics

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8

{
  "system" : {
    "minorPageFaults" : 23048,
    "majorPageFaults" : 2604,
    "userTime" : 0.644947,
    "systemTime" : 0.302917,
    "numberOfThreads" : 16,
    "residentSize" : 84783104,
    "virtualSize" : 5005017088
  },
  "client" : {
    "httpConnections" : 1,
    "connectionTime" : {
      "sum" : 0,
      "count" : 0,
      "counts" : [
        0,
        0,
        0,
        0
      ]
    },
    "totalTime" : {
      "sum" : 0.22414636611938477,
      "count" : 12,
      "counts" : [
```

```
        6,
        5,
        1,
        0,
        0,
        0,
        0
      ]
    },
    "requestTime" : {
      "sum" : 0.22205686569213867,
      "count" : 12,
      "counts" : [
        6,
        5,
        1,
        0,
        0,
        0,
        0
      ]
    },
    "queueTime" : {
      "sum" : 0.00022125244140625,
      "count" : 11,
      "counts" : [
        11,
        0,
        0,
        0,
        0,
        0,
        0
      ]
    },
    "bytesSent" : {
      "sum" : 4160,
      "count" : 12,
      "counts" : [
        3,
        8,
        1,
        0,
        0,
```

```
        0
      ]
    },
    "bytesReceived" : {
      "sum" : 2300,
      "count" : 12,
      "counts" : [
        10,
        2,
        0,
        0,
        0,
        0
      ]
    }
  },
  "error" : false,
  "code" : 200
}
```

GET /_admin/statistics-description,statistics description

Returns a description of the statistics returned by `/_admin/statistics`. The returned objects contains a list of statistics groups in the attribute `groups` and a list of statistics figures in the attribute `figures`.

A statistics group is described by

- `group`: The identifier of the group.
- `name`: The name of the group.
- `description`: A description of the group.

A statistics figure is described by

- `group`: The identifier of the group to which this figure belongs.
- `identifier`: The identifier of the figure. It is unique within the group.
- `name`: The name of the figure.
- `description`: A description of the group.
- `type`: Either `current`, `accumulated`, or `distribution`.
- `cuts`: The distribution vector.
- `units`: Units in which the figure is measured.

Description was returned successfully.

```
unix> curl --dump - http://localhost:8529/_admin/
   statistics-description

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8

{
  "groups" : [
    {
      "group" : "system",
      "name" : "Process Statistics",
      "description" : "Statistics about the ArangoDB
         process"
    },
    {
```

```
      "group" : "client",
      "name" : "Client Statistics",
      "description" : "Statistics about the clients
         connecting to the server."
    }
  ],
  "figures" : [
    {
      "group" : "system",
      "identifier" : "userTime",
      "name" : "User Time",
      "description" : "Amount of time that this
         process has been scheduled in user mode,
         measured in clock ticks divided by sysconf(
         _SC_CLK_TCK) aka seconds.",
      "type" : "accumulated",
      "units" : "seconds"
    },
    {
      "group" : "system",
      "identifier" : "systemTime",
      "name" : "System Time",
      "description" : "Amount of time that this
         process has been scheduled in kernel mode,
         measured in clock ticks divided by sysconf(
         _SC_CLK_TCK) aka seconds.",
      "type" : "accumulated",
      "units" : "seconds"
    },
    {
      "group" : "system",
      "identifier" : "numberOfThreads",
      "name" : "Number of Threads",
      "description" : "Number of threads in this
         process.",
      "type" : "current",
      "units" : "number"
    },
    {
      "group" : "system",
      "identifier" : "residentSize",
      "name" : "Resident Set Size",
      "description" : "The number of pages the process
          has in real memory. This is just the pages
```

```
        which count toward text, data, or stack space
        . This does not include pages which have not
        been demand-loaded in, or which are swapped
        out.",
      "type" : "current",
      "units" : "bytes"
    },
    {
      "group" : "system",
      "identifier" : "virtualSize",
      "name" : "Virtual Memory Size",
      "description" : "The size of the virtual memory
        the process is using.",
      "type" : "current",
      "units" : "bytes"
    },
    {
      "group" : "system",
      "identifier" : "minorPageFaults",
      "name" : "Minor Page Faults",
      "description" : "The number of minor faults the
        process has made which have not required
        loading a memory page from disk.",
      "type" : "accumulated",
      "units" : "number"
    },
    {
      "group" : "system",
      "identifier" : "majorPageFaults",
      "name" : "Major Page Faults",
      "description" : "The number of major faults the
        process has made which have required loading
        a memory page from disk.",
      "type" : "accumulated",
      "units" : "number"
    },
    {
      "group" : "client",
      "identifier" : "httpConnections",
      "name" : "HTTP Client Connections",
      "description" : "The number of http connections
        that are currently open.",
      "type" : "current",
      "units" : "number"
```

```
    },
    {
      "group" : "client",
      "identifier" : "totalTime",
      "name" : "Total Time",
      "description" : "Total time needed to answer a
         request.",
      "type" : "distribution",
      "cuts" : [
        0.01,
        0.05,
        0.1,
        0.2,
        0.5,
        1
      ],
      "units" : "seconds"
    },
    {
      "group" : "client",
      "identifier" : "requestTime",
      "name" : "Request Time",
      "description" : "Request time needed to answer a
          request.",
      "type" : "distribution",
      "cuts" : [
        0.01,
        0.05,
        0.1,
        0.2,
        0.5,
        1
      ],
      "units" : "seconds"
    },
    {
      "group" : "client",
      "identifier" : "queueTime",
      "name" : "Queue Time",
      "description" : "Queue time needed to answer a
         request.",
      "type" : "distribution",
      "cuts" : [
        0.01,
```

```
        0.05,
        0.1,
        0.2,
        0.5,
        1
      ],
      "units" : "seconds"
    },
    {
      "group" : "client",
      "identifier" : "bytesSent",
      "name" : "Bytes Sent",
      "description" : "Bytes sents for a request.",
      "type" : "distribution",
      "cuts" : [
        250,
        1000,
        2000,
        5000,
        10000
      ],
      "units" : "bytes"
    },
    {
      "group" : "client",
      "identifier" : "bytesReceived",
      "name" : "Bytes Received",
      "description" : "Bytes receiveds for a request
         .",
      "type" : "distribution",
      "cuts" : [
        250,
        1000,
        2000,
        5000,
        10000
      ],
      "units" : "bytes"
    },
    {
      "group" : "client",
      "identifier" : "connectionTime",
      "name" : "Connection Time",
      "description" : "Total connection time of a
```

```
         client.",
      "type" : "distribution",
      "cuts" : [
        0.1,
        1,
        60
      ],
      "units" : "seconds"
    }
  ],
  "error" : false,
  "code" : 200
}
```

Chapter 19

HTTP Interface for User Management

19.1 User Management

This is an introduction to ArangoDB's Http interface for managing users.

The interface provides a simple means to add, update, and remove users. All users managed through this interface will be stored in the system collection `_users`.

This specialised interface intentionally does not provide all functionality that is available in the regular document REST API.

Operations on users may become more restricted than regular document operations, and extra privilege and security security checks may be introduced in the future for this interface.

POST /_api/user,creates user

POST /_api/user

The following data need to be passed in a JSON representation in the body of the POST request:

- `username`: The name of the user as a string. This is mandatory.
- `passwd`: The user password as a string. If no password is specified, the empty string will be used.
- `active`: an optional flag that specifies whether the user is active. If not specified, this will default to `true`.
- `extra`: an optional JSON object with arbitrary extra data about the user.

If the user can be added by the server, the server will respond with `HTTP 201`.

In case of success, the returned JSON object has the following properties:

- `error`: boolean flag to indicate that an error occurred (`false` in this case)
- `code`: the HTTP status code

If the JSON representation is malformed or mandatory data is missing from the request, the server will respond with `HTTP 400`.

The body of the response will contain a JSON object with additional error details. The object has the following attributes:

- `error`: boolean flag to indicate that an error occurred (`true` in this case)
- `code`: the HTTP status code
- `errorNum`: the server error number
- `errorMessage`: a descriptive error message

PUT /_api/user,replaces user

PUT /_api/user/`username`

Replaces the data of an existing user. The name of an existing user must be specified in `username`.

The following data can to be passed in a JSON representation in the body of the POST request:

- `passwd`: The user password as a string. Specifying a password is mandatory, but the empty string is allowed for passwords.
- `active`: an optional flag that specifies whether the user is active. If not specified, this will default to `true`.
- `extra`: an optional JSON object with arbitrary extra data about the user.

If the user can be replaced by the server, the server will respond with `HTTP 200`.

In case of success, the returned JSON object has the following properties:

- `error`: boolean flag to indicate that an error occurred (`false` in this case)
- `code`: the HTTP status code

If the JSON representation is malformed or mandatory data is missing from the request, the server will respond with `HTTP 400`. If the specified user does not exist, the server will respond with `HTTP 404`.

The body of the response will contain a JSON object with additional error details. The object has the following attributes:

- `error`: boolean flag to indicate that an error occurred (`true` in this case)
- `code`: the HTTP status code
- `errorNum`: the server error number
- `errorMessage`: a descriptive error message

PATCH /_api/user,updates user

PATCH /_api/user/username

Partially updates the data of an existing user. The name of an existing user must be specified in username.

The following data can be passed in a JSON representation in the body of the POST request:

- passwd: The user password as a string. Specifying a password is optional. If not specified, the previously existing value will not be modified.
- active: an optional flag that specifies whether the user is active. If not specified, the previously existing value will not be modified.
- extra: an optional JSON object with arbitrary extra data about the user. If not specified, the previously existing value will not be modified.

If the user can be updated by the server, the server will respond with HTTP 200.

In case of success, the returned JSON object has the following properties:

- error: boolean flag to indicate that an error occurred (false in this case)
- code: the HTTP status code

If the JSON representation is malformed or mandatory data is missing from the request, the server will respond with HTTP 400. If the specified user does not exist, the server will respond with HTTP 404.

The body of the response will contain a JSON object with additional error details. The object has the following attributes:

- error: boolean flag to indicate that an error occurred (true in this case)
- code: the HTTP status code
- errorNum: the server error number
- errorMessage: a descriptive error message

DELETE /_api/user,removes a user

DELETE /_api/user/`username`

Removes an existing user, identified by `username`.

If the user can be removed by the server, the server will respond with `HTTP 202`.

In case of success, the returned JSON object has the following properties:

- `error`: boolean flag to indicate that an error occurred (`false` in this case)
- `code`: the HTTP status code

If the specified user does not exist, the server will respond with `HTTP 404`.

The body of the response will contain a JSON object with additional error details. The object has the following attributes:

- `error`: boolean flag to indicate that an error occurred (`true` in this case)
- `code`: the HTTP status code
- `errorNum`: the server error number
- `errorMessage`: a descriptive error message

GET /_api/user,fetches a user

GET /_api/user/`username`

Fetches data about the specified user.

The call will return a JSON document with at least the following attributes on success:

- `username`: The name of the user as a string.
- `active`: an optional flag that specifies whether the user is active.
- `extra`: an optional JSON object with arbitrary extra data about the user.

Chapter 20

HTTP Interface for Miscellaneous functions

This is an overview of ArangoDB's HTTP interface for miscellaneous functions.

GET /_admin/version,returns the server version number

GET /_admin/version

Returns an object containing the server name in the `server` attribute, and the current server version in the `version` attribute.

```
> curl -X GET --dump - http://localhost:8529/_admin/
   version

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8

{
  "server": "arango",
  "version": "1.3.devel"
}
```

GET /_admin/version?details=true

If the optional URL parameter `details` is set to `true`, then more server version details are returned in the `details` attribute. The details are returned as pairs of attribute name and value, which are all strings. The number of attributes may vary, depending on the server built and configuration.

```
> curl -X GET --dump - http://localhost:8529/_admin/
   version?details=true

HTTP/1.1 200 OK
content-type: application/json; charset=utf-8

{
  "server": "arango",
  "version": "1.3.devel"
  "details": {
    "build-date": "Mar  8 2013 22:06:49",
    "configure": "'./configure' '--enable-maintainer-
       mode' '--enable-relative' '--enable-all-in-one-
       icu' '--enable-all-in-one-v8'","icu-version
       ":"49.1.2",
    "libev-version": "4.11",
    "openssl-version": "OpenSSL 1.0.1 14 Mar 2012",
    "repository-version": "heads/devel-0-
       gdd446b51dd713656d0f3dc50e94e3ea88cd72a86",
    "server-version": "1.3.devel",
    "v8-version":"3.16.14.1"
  }
```

```
}
```

GET /_admin/time,returns the system time

GET /_admin/time

The call returns an object with the attribute `time`. This contains the current system time as a Unix timestamp with microsecond precision.

GET /_admin/echo,returns the current request

GET /_admin/echo

The call returns an object with the following attributes:

- `headers`: a list of HTTP headers received
- `requestType`: the HTTP request method (e.g. GET)
- `parameters`: list of URL parameters received

Chapter 21

HTTP Handling in ArangoDB

ArangoDB will always respond to client requests with HTTP 1.1. Clients should therefore support HTTP version 1.1.

21.1 Keep-Alive and Authentication

ArangoDB supports HTTP keep-alive. If the client does not send a `Connection` header in its request, and the client uses HTTP version 1.1, ArangoDB will assume the client wants to keep alive the connection. If clients do not wish to use the keep-alive feature, they should explicitly indicate that by sending a `Connection: Close` HTTP header in the request.

ArangoDB will close connections automatically for clients that send requests using HTTP 1.0, except if they send an `Connection: Keep-Alive` header.

The default Keep-Alive timeout can be specified at server start using the `\-\-server.keep\-alive\-timeout` parameter.

Client authentication is done by using the `Authorization` HTTP header. ArangoDB supports Basic authentication.

Authentication is optional. To enforce authentication for incoming requested, the server must be started with the option `\-\-server.disable-authentication`. Please note that requests using the HTTP OPTIONS method will be answered by ArangoDB in any case, even if no authentication data is sent by the client or if the authentication data is wrong. This is required for handling CORS preflight requests (see **Cross Origin Resource Sharing (CORS) requests** (p. 188)). The response to an HTTP OPTIONS request will be generic and not expose any private data.

21.2 Error Handling

The following should be noted about how ArangoDB handles client errors in its HTTP layer:

- ArangoDB will reject client requests with a negative value in the `Content-Length` request header with `HTTP 411` (Length Required).
- if the client sends a `Content-Length` header with a value bigger than 0 for an HTTP GET, HEAD, or DELETE request, ArangoDB will process the request, but will write a warning to its log file.
- when the client sends a `Content-Length` header that has a value that is lower than the actual size of the body sent, ArangoDB will respond with `HTTP 400` (Bad Request).
- if clients send a `Content-Length` value bigger than the actual size of the body of the request, ArangoDB will wait for about 90 seconds for the client to complete its request. If the client does not send the remaining body data within this time, ArangoDB will close the connection. Clients should avoid sending such malformed requests as they will make ArangoDB block waiting for more data to arrive.
- when clients send a body or a `Content-Length` value bigger than the maximum allowed value (512 MB), ArangoDB will respond with `HTTP 413` (Request Entity Too Large).
- if the overall length of the HTTP headers a client sends for one request exceeds the maximum allowed size (1 MB), the server will fail with `HTTP 431` (Request Header Fields Too Large).
- if clients request a HTTP method that is not supported by the server, ArangoDB will return with `HTTP 405` (Method Not Allowed). ArangoDB offers general support for the following HTTP methods:
  - GET
  - POST
  - PUT
  - DELETE
  - HEAD
  - PATCH

  - OPTIONS

Please note that not all server actions allow using all of these HTTP methods. You should look up up the supported methods for each method you intend to use in the manual.

Requests using any other HTTP method (such as for example CONNECT, TRACE etc.) will be rejected by ArangoDB.

21.3 Cross Origin Resource Sharing (CORS) requests

ArangoDB will automatically handle CORS requests as follows:

- when the client sends an `Origin` HTTP header, ArangoDB will return a header `access-control-allow-origin` containing the value the client sent in the `Origin` header.

- for non-trivial CORS requests, clients may issue a preflight request via an additional HTTP OPTIONS request. ArangoDB will automatically answer such preflight HTTP OPTIONS requests with an HTTP 200 response with an empty body. ArangoDB will return the following headers in the response:

  - `access-control-allow-origin`: will contain the value that the client provided in the `Origin` header of the request

  - `access-control-allow-methods`: will contain a list of all HTTP methods generally supported by ArangoDB. This list does not depend on the URL the client requested and is the same for all CORS requests.

  - `access-control-allow-headers`: will contain exactly the value that the client has provided in the `Access-Control-Request-Header` header of the request. This header will only be returned if the client has specified the header in the request. ArangoDB will send back the original value without further validation.

  - `access-control-max-age`: will return a cache lifetime for the preflight response as determined by ArangoDB.

- any `access-control-allow-credentials` header sent by the client is ignored by ArangoDB its value is not `true`. If a client sends a header value of `true`, ArangoDB will return the header `access-control-allow-credentials: true`, too.

Note that CORS preflight requests will probably not send any authentication data with them. One of the purposes of the preflight request is to check whether the server accepts authentication or not.

A consequence of this is that ArangoDB will allow requests using the HTTP OPTIONS method without credentials, even when the server is run with authentication enabled.

The response to the HTTP OPTIONS request will however be a generic response that will not expose any private data and thus can be considered "safe" even without credentials.

Chapter 22

Naming Conventions in ArangoDB

The following naming conventions should be followed by users when creating collections and documents in ArangoDB.

22.1 Collection Names

Users can pick names for their collections as desired, provided the following naming constraints are not violated:

- Collection names must only consist of the letters `a` to `z` (both in lower and upper case), the numbers `0` to `9`, the the underscore (`_`), or the dash (`-`) symbol. This also means that any non-ASCII collection names are not allowed.
- Collection names must start with a letter (not a number, the underscore or the dash character). Collection names starting with an underscore are considered to be system collections that are for ArangoDB's internal use only. System collection names should not be used by end users for their own collections.
- The maximum allowed length of a collection name is 64 bytes.
- Collection names are case-sensitive.

22.2 Document Keys

Users can define their own keys for documents they save. The document key will be saved along with a document in the `_key` attribute. Users can pick key values as required, provided that the values conform to the following restrictions:

- the key must be at least 1 byte and at most 254 bytes long. empty keys are disallowed when specified (though it may be valid to completely omit the `_key` attribute from a document)
- it must consist of the letters a-z (lower or upper case), the digits 0-9, the underscore (_), dash (-), or colon (:) characters only
  - any other characters, especially multi-byte sequences, whitespace or punctuation characters cannot be used inside key values
- the key must be unique within the collection it is used

Keys are case-sensitive, i.e. `myKey` and `MyKEY` are considered different keys.

Specifiying a document key is optional when creating new documents. If no document key is specified by the user, ArangoDB will create the document key itself as each document is required to have a key.

There are no guarantees about the format and pattern of auto-generated document keys other than the above restrictions. Clients should therefore treat auto-generated document keys as opaque values and not rely on their format.

22.3 Attribute Names

Users can pick attribute names for document attributes as desired, provided the following attribute naming constraints are not violated:

- Attribute names starting with an underscore are considered to be system attributes for ArangoDB's internal use. Such attribute names are already used by ArangoDB for special purposes, e.g. `_id` is used to contain a document's handle, `_key` is used to contain a document's user-defined key, and `_rev` is used to contain the document's revision number. In edge collections, the `_from` and `_to` attributes are used to reference other documents.

  More system attributes may be added in the future without further notice so end users should not use attribute names starting with an underscore for their own attributes.

- Attribute names should not start with the at-mark (`\@`). The at-mark at the start of attribute names is reserved in ArangoDB for future use cases.
- Theoretically, attribute names can include punctuation and special characters as desired, provided the name is a valid UTF-8 string. For maximum portability, special characters should be avoided though. For example, attribute names may contain the dot symbol, but the dot has a special meaning

in Javascript and also in AQL, so when using such attribute names in one of these languages, the attribute name would need to be quoted by the end user. This will work but requires more work so it might be better to use attribute names which don't require any quoting/escaping in all languages used. This includes languages used by the client (e.g. Ruby, PHP) if the attributes are mapped to object members there.

- ArangoDB does not enforce a length limit for attribute names. However, long attribute names may use more memory in result sets etc. Therefore the use of long attribute names is discouraged.
- As ArangoDB saves document attribute names separate from the actual document attribute value data, the combined length of all attribute names for a document must fit into an ArangoDB shape structure. The maximum combined names length is variable and depends on the number and data types of attributes used.
- Attribute names are case-sensitive.
- Attributes with empty names (the empty string) and attributes with names that start with an underscore and don't have a special meaning (system attributes) are removed from the document when saving it.

  When the document is later requested, it will be returned without these attributes. For example, if this document is saved

  ```
  { "a" : 1, "" : 2, "_test" : 3, "b": 4 }
  ```

  and later requested, it will be returned like this:

  ```
  { "a" : 1, "b": 4 }
  ```

Chapter 23

Error codes and meanings

The following errors might be raised when running ArangoDB:

- 0: `no error` No error has occurred.
- 1: `failed` Will be raised when a general error occurred.
- 2: `system error` Will be raised when operating system error occurred.
- 3: `out of memory` Will be raised when there is a memory shortage.
- 4: `internal error` Will be raised when an internal error occurred.
- 5: `illegal number` Will be raised when an illegal representation of a number was given.
- 6: `numeric overflow` Will be raised when a numeric overflow occurred.
- 7: `illegal option` Will be raised when an unknown option was supplied by the user.
- 8: `dead process identifier` Will be raised when a PID without a living process was found.
- 9: `not implemented` Will be raised when hitting an unimplemented feature.
- 10: `bad parameter` Will be raised when the parameter does not fulfill the requirements.
- 11: `forbidden` Will be raised when you are missing permission for the operation.
- 12: `out of memory in mmap` Will be raised when there is a memory shortage.

- 13: `csv is corrupt` Will be raised when encountering a corrupt csv line.
- 14: `file not found` Will be raised when a file is not found.
- 15: `cannot write file` Will be raised when a file cannot be written.
- 16: `cannot overwrite file` Will be raised when an attempt is made to overwrite an existing file.
- 17: `type error` Will be raised when a type error is unencountered.
- 18: `lock timeout` Will be raised when there's a timeout waiting for a lock.
- 400: `bad parameter` Will be raised when the HTTP request does not fulfill the requirements.
- 403: `forbidden` Will be raised when the operation is forbidden.
- 404: `not found` Will be raised when an URI is unknown.
- 405: `method not supported` Will be raised when an unsupported HTTP method is used for an operation.
- 500: `internal server error` Will be raised when an internal server is encountered.
- 600: `invalid JSON object` Will be raised when a string representation of a JSON object is corrupt.
- 601: `superfluous URL suffices` Will be raised when the URL contains superfluous suffices.
- 1000: `illegal state` Internal error that will be raised when the datafile is not in the required state.
- 1001: `could not shape document` Internal error that will be raised when the shaper encountered a problem.
- 1002: `datafile sealed` Internal error that will be raised when trying to write to a datafile.
- 1003: `unknown type` Internal error that will be raised when an unknown collection type is encountered.
- 1004: `ready only` Internal error that will be raised when trying to write to a read-only datafile or collection.
- 1005: `duplicate identifier` Internal error that will be raised when a identifier duplicate is detected.

- 1006: `datafile unreadable` Internal error that will be raised when the datafile is unreadable.
- 1100: `corrupted datafile` Will be raised when a corruption is detected in a datafile.
- 1101: `illegal parameter file` Will be raised if a parameter file is corrupted.
- 1102: `corrupted collection` Will be raised when a collection contains one or more corrupted datafiles.
- 1103: `mmap failed` Will be raised when the system call mmap failed.
- 1104: `filesystem full` Will be raised when the filesystem is full.
- 1105: `no journal` Will be raised when a journal cannot be created.
- 1106: `cannot create/rename datafile because it already exists` Will be raised when the datafile cannot be created or renamed because a file of the same name already exists.
- 1107: `database is locked` Will be raised when the database is locked by a different process.
- 1108: `cannot create/rename collection because directory already exists` Will be raised when the collection cannot be created because a directory of the same name already exists.
- 1109: `msync failed` Will be raised when the system call msync failed.
- 1200: `conflict` Will be raised when updating or deleting a document and a conflict has been detected.
- 1201: `wrong path for database` Will be raised when a non-existing directory was specified as path for the database.
- 1202: `document not found` Will be raised when a document with a given identifier or handle is unknown.
- 1203: `collection not found` Will be raised when a collection with a given identifier or name is unknown.
- 1204: `parameter 'collection' not found` Will be raised when the collection parameter is missing.
- 1205: `illegal document handle` Will be raised when a document handle is corrupt.

- 1206: `maixmal size of journal too small` Will be raised when the maximal size of the journal is too small.
- 1207: `duplicate name` Will be raised when a name duplicate is detected.
- 1208: `illegal name` Will be raised when an illegal name is detected.
- 1209: `no suitable index known` Will be raised when no suitable index for the query is known.
- 1210: `unique constraint violated` Will be raised when there is a unique constraint violation.
- 1211: `geo index violated` Will be raised when a illegale coordinate is used.
- 1212: `index not found` Will be raised when an index with a given identifier is unknown.
- 1213: `cross collection request not allowed` Will be raised when a cross-collection is requested.
- 1214: `illegal index handle` Will be raised when a index handle is corrupt.
- 1215: `cap constraint already defined` Will be raised when a cap constraint was already defined.
- 1216: `document too large` Will be raised when the document cannot fit into any datafile because of it is too large.
- 1217: `collection must be unloaded` Will be raised when a collection should be unloaded, but has a different status.
- 1218: `collection type invalid` Will be raised when an invalid collection type is used in a request.
- 1219: `validator failed` Will be raised when the validation of an attribute of a structure failed.
- 1220: `parser failed` Will be raised when the parsing of an attribute of a structure failed.
- 1221: `illegal document key` Will be raised when a document key is corrupt.
- 1222: `unexpected document key` Will be raised when a user-defined document key is supplied for collections with auto key generation.

- 1223: `index needs resizing` Will be raised when an index is full and should be resized to contain more data.
- 1224: `database directory not writable` Will be raised when the database directory is not writable for the current user.
- 1225: `out of keys` Will be raised when a key generator runs out of keys.
- 1226: `missing document key` Will be raised when a document key is missing.
- 1227: `invalid document type` Will be raised when there is an attempt to create a document with an invalid type.
- 1300: `datafile full` Will be raised when the datafile reaches its limit.
- 1500: `query killed` Will be raised when a running query is killed by an explicit admin command.
- 1501: `%s` Will be raised when query is parsed and is found to be syntactially invalid.
- 1502: `query is empty` Will be raised when an empty query is specified.
- 1503: `runtime error '%s'` Will be raised when a runtime error is caused by the query.
- 1504: `number out of range` Will be raised when a number is outside the expected range.
- 1510: `variable name '%s' has an invalid format` Will be raised when an invalid variable name is used.
- 1511: `variable '%s' is assigned multiple times` Will be raised when a variable gets re-assigned in a query.
- 1512: `unknown variable '%s'` Will be raised when an unknown variable is used or the variable is undefined the context it is used.
- 1521: `unable to read-lock collection %s` Will be raised when a read lock on the collection cannot be acquired.
- 1522: `too many collections` Will be raised when the number of collections in a query is beyond the allowed value.
- 1530: `document attribute '%s' is assigned multiple times` Will be raised when a document attribute is re-assigned.
- 1540: `usage of unknown function '%s()'` Will be raised when an undefined function is called.

- 1541: `invalid number of arguments for function '%s()'` Will be raised when the number of arguments used in a function call does not match the expected number of arguments for the function.
- 1542: `invalid argument type used in call to function '%s()'` Will be raised when the type of an argument used in a function call does not match the expected argument type.
- 1543: `invalid regex argument value used in call to function '%s()'` Will be raised when an invalid regex argument value is used in a call to a function that expects a regex.
- 1550: `invalid structure of bind parameters` Will be raised when the structure of bind parameters passed has an unexpected format.
- 1551: `no value specified for declared bind parameter '%s'` Will be raised when a bind parameter was declared in the query but the query is being executed with no value for that parameter.
- 1552: `bind parameter '%s' was not declared in the query` Will be raised when a value gets specified for an undeclared bind parameter.
- 1553: `bind parameter '%s' has an invalid value or type` Will be raised when a bind parameter has an invalid value or type.
- 1560: `invalid logical value` Will be raised when a non-boolean value is used in a logical operation.
- 1561: `invalid arithmetic value` Will be raised when a non-numeric value is used in an arithmetic operation.
- 1562: `division by zero` Will be raised when there is an attempt to divide by zero.
- 1563: `list expected` Will be raised when a non-list operand is used for an operation that expects a list argument operand.
- 1569: `FAIL(%s) called` Will be raised when the function FAIL() is called from inside a query.
- 1570: `no suitable geo index found for geo restriction on '%s'` Will be raised when a geo restriction was specified but no suitable geo index is found to resolve it.
- 1571: `no suitable fulltext index found for fulltext query on '%s'` Will be raised when a fulltext query is performed on a collection without a suitable fulltext index.

- 1580: `invalid user function name` Will be raised when a user function with an invalid name is registered.
- 1581: `invalid user function code` Will be raised when a user function is registered with invalid code.
- 1582: `user function '%s()' not found` Will be raised when a user function is accessed but not found.
- 1600: `cursor not found` Will be raised when a cursor is requested via its id but a cursor with that id cannot be found.
- 1650: `internal transaction error` Will be raised when a wrong usage of transactions is detected. this is an internal error and indicates a bug in ArangoDB.
- 1651: `nested transactions detected` Will be raised when transactions are nested.
- 1652: `unregistered collection used in transaction` Will be raised when a collection is used in the middle of a transaction but was not registered at transaction start.
- 1653: `disallowed operation inside transaction` Will be raised when a disallowed operation is carried out in a transaction.
- 1700: `invalid user name` Will be raised when an invalid user name is used
- 1701: `invalid password` Will be raised when an invalid password is used
- 1702: `duplicate user` Will be raised when a user name already exists
- 1703: `user not found` Will be raised when a user name is updated that does not exist
- 1750: `application not found` Will be raised when an application is not found or not present in the specified version.
- 1751: `invalid application name` Will be raised when an invalid application name is specified.
- 1752: `invalid mount` Will be raised when an invalid mount is specified.
- 1753: `application download failed` Will be raised when an application download from the central repository failed.
- 1800: `invalid key declaration` Will be raised when an invalid key specification is passed to the server

- 1801: `key already exists` Will be raised when a key is to be created that already exists
- 1802: `key not found` Will be raised when the specified key is not found
- 1803: `key is not unique` Will be raised when the specified key is not unique
- 1804: `key value not changed` Will be raised when updating the value for a key does not work
- 1805: `key value not removed` Will be raised when deleting a key/-value pair does not work
- 1806: `missing value` Will be raised when the value is missing
- 1901: `invalid graph` Will be raised when an invalid name is passed to the server
- 1902: `could not create graph` Will be raised when an invalid name, vertices or edges is passed to the server
- 1903: `invalid vertex` Will be raised when an invalid vertex id is passed to the server
- 1904: `could not create vertex` Will be raised when the vertex could not be created
- 1905: `could not change vertex` Will be raised when the vertex could not be changed
- 1906: `invalid edge` Will be raised when an invalid edge id is passed to the server
- 1907: `could not create edge` Will be raised when the edge could not be created
- 1908: `could not change edge` Will be raised when the edge could not be changed
- 1951: `invalid session` Will be raised when an invalid session id is passed to the server
- 1952: `could not create session` Will be raised when the session could not be created
- 1953: `could not change session` Will be raised when session data could not be changed

- 1961: `invalid form` Will be raised when an invalid form id is passed to the server
- 1962: `could not create form` Will be raised when the form could not be created
- 1963: `could not change form` Will be raised when form data could not be changed
- 2000: `unknown client error` This error should not happen.
- 2001: `could not connect to server` Will be raised when the client could not connect to the server.
- 2002: `could not write to server` Will be raised when the client could not write data.
- 2003: `could not read from server` Will be raised when the client could not read data.
- 3100: `priority queue insert failure` Will be raised when an attempt to insert a document into a priority queue index fails for some reason.
- 3110: `priority queue remove failure` Will be raised when an attempt to remove a document from a priority queue index fails for some reason.
- 3111: `priority queue remove failure - item missing in index` Will be raised when an attempt to remove a document from a priority queue index fails when document can not be located within the index.
- 3312: `(non-unique) hash index insert failure - document duplicated in index` Will be raised when an attempt to insert a document into a non-unique hash index fails due to the fact that document is duplicated within that index.
- 3313: `(non-unique) skiplist index insert failure - document duplicated in index` Will be raised when an attempt to insert a document into a non-unique skiplist index fails due to the fact that document is duplicated within that index.
- 3200: `hash index insertion warning - attribute missing in document` Will be raised when an attempt to insert a document into a hash index is caused by the document not having one or more attributes which are required by the hash index.
- 3202: `hash index update warning - attribute missing in revised document` Will be raised when an attempt to update a document results in the revised document not having one or more attributes which are required by the hash index.

- 3211: `hash index remove failure - item missing in index` Will be raised when an attempt to remove a document from a hash index fails when document can not be located within that index.
- 3300: `skiplist index insertion warning - attribute missing in document` Will be raised when an attempt to insert a document into a skiplist index is caused by in the document not having one or more attributes which are required by the skiplist index.
- 3302: `skiplist index update warning - attribute missing in revised document` Will be raised when an attempt to update a document results in the revised document not having one or more attributes which are required by the skiplist index.
- 3311: `skiplist index remove failure - item missing in index` Will be raised when an attempt to remove a document from a skiplist index fails when document can not be located within that index.
- 3400: `bitarray index insertion warning - attribute missing in document` Will be raised when an attempt to insert a document into a bitarray index is caused by in the document not having one or more attributes which are required by the bitarray index.
- 3402: `bitarray index update warning - attribute missing in revised document` Will be raised when an attempt to update a document results in the revised document not having one or more attributes which are required by the bitarray index.
- 3411: `bitarray index remove failure - item missing in index` Will be raised when an attempt to remove a document from a bitarray index fails when document can not be located within that index.
- 3413: `bitarray index insert failure - document attribute value unsupported in index` Will be raised when an attempt to insert a document into a bitarray index fails due to the fact that one or more values for an index attribute is not supported within that index.
- 3415: `bitarray index creation failure - one or more index attributes are duplicated.` Will be raised when an attempt to create an index with two or more index attributes repeated.
- 3417: `bitarray index creation failure - one or more index attribute values are duplicated.` Will be raised when an attempt to create an index with two or more index attribute values repeated.
- 10000: `element not inserted into structure, because key already exists` Will be returned if the element was not insert because the key already exists.

- 10001: `element not inserted into structure, because it already exists` Will be returned if the element was not insert because it already exists.
- 10002: `key not found in structure` Will be returned if the key was not found in the structure.
- 10003: `element not found in structure` Will be returned if the element was not found in the structure.

Chapter 24

Glossary

24.1 Glossary

Collection: A collection consists of documents. It is uniquely identified by the server via its collection identifier. It also has a unique name that clients should use to identify and access it. Collections have a type that is specified by the user when the collection is created. There currently are *document* and *edge* collections. The default type is *document*.

Collection Identifier: A collection identifier identifies a collection in a database. It is a string value and is unique within the database. Up to including ArangoDB 1.1, the collection identifier has been a client's primary means to access collections. Starting with ArangoDB 1.2, clients should instead use a collection's unique name to access a collection instead of its identifier.

ArangoDB currently uses 64bit unsigned integer values to maintain collection ids internally. When returning collection ids to clients, ArangoDB will put them into a string to ensure the collection id is not clipped by clients that do not support big integers. Clients should treat the collection ids returned by ArangoDB as opaque strings when they store or use it locally.

Note: collection ids have been returned as integers up to including ArangoDB 1.1

Collection Name: A collection name identifies a collection in a database. It is a string and is unique within the database. Unlike the collection identifier it is supplied by the creator of the collection. The collection name must consist of letters, digits and the characters _ (underscore), - (dash), and : (colon). Please refer to **Naming Conventions in ArangoDB** (p. 190) for more information on valid collection names.

Document: Documents in ArangoDB are JSON objects. These objects can be nested (to any depth) and may contains lists. Each document is uniquely identified

by its document handle.

Document Etag: The document revision enclosed in double quotes. The revision is returned by several HTTP API methods in the `Etag` HTTP header.

Document Handle: A document handle uniquely identifies a document in the database. It is a string and consists of a collection name and a document key separated by `/`.

Document Key: A document key uniquely identifies a document in a given collection. It can and should be used by clients when specific documents are searched. Document keys are stored in the `_key` attribute of documents. The key values are automatically indexed by ArangoDB so looking up a document by its key is regularly a fast operation. The `_key` value of a document is immutable once the document has been created.

By default, ArangoDB will auto-generate a document key if no `_key` attribute is specified, and use the user-specified `_key` otherwise.

This behavior can be changed on a per-collection level by creating collections with the `keyOptions` attribute.

Using `keyOptions` it is possible to disallow user-specified keys completely, or to force a specific regime for auto-generating the `_key` values.

Document Revision: As ArangoDB supports MVCC, documents can exist in more than one revision. The document revision is the MVCC token used to identify a particular revision of a document. It is a string value currently containing an integer number and is unique within the list of document revisions for a single document. Document revisions can be used to conditionally update, replace or delete documents in the database. In order to find a particular revision of a document, you need the document handle and the document revision.

ArangoDB currently uses 64bit unsigned integer values to maintain document revisions internally. When returning document revisions to clients, ArangoDB will put them into a string to ensure the revision id is not clipped by clients that do not support big integers. Clients should treat the revision id returned by ArangoDB as an opaque string when they store or use it locally. This will allow ArangoDB to change the format of revision ids later if this should be required. Clients can use revisions ids to perform simple equality/non-equality comparisons (e.g. to check whether a document has changed or not), but they should not use revision ids to perform greater/less than comparisions with them to check if a document revision is older than one another, even if this might work for some cases.

Note: revision ids have been returned as integers up to including ArangoDB 1.1

Edge: Edges in ArangoDB are special documents. In addition to the internal attributes `_key`, `_id` and `_rev`, they have two attributes `_from` and `_to`, which contain document handles, namely the start-point and the end-point of the edge.

`_from` and `_to` contain document handles, e.g.:

```
{ "_from" : "myvertices/doc1", "_to" : "myvertices/doc2", ... }
```

The values of `_from` and `_to` are immutable once saved.

Edge Collection: Edge collections are special collection that store edge documents. Edge documents are connection documents that reference other documents. The type of a collection must be specified when a collection is created and cannot be changed afterwards.

Index: Indexes are used to allow fast access to documents. For each collection there is always the primary index which is a hash index for the document key. This index cannot be dropped or changed.

Edge collections will also have an automatically created edges index, which cannot be modified.

Most user-land indexes can be created by defining the names of the attributes which should be indexed. Some index types allow indexing just one attribute (e.g. fulltext index) whereas other index types allow indexing multiple attributes.

Indexing system attributes such as `_id`, `_key`, `_from`, and `_to` is not supported by any index type. Manually creating an index that relies on any of these attributes is unsupported.

Index Handle: An index handle uniquely identifies an index in the database. It is a string and consists of a collection name and an index identifier separated by `/`.

Edges Index: An edges index is automatically created for edge collections. It contains connections between vertex documents and is invoked when the connecting edges of a vertex are queried. There is no way to explicitly create or delete edge indexes.

Fulltext Index: A fulltext index can be used to find words, or prefixes of words inside documents. A fulltext index can be set on one attribute only, and will index all words contained in documents that have a textual value in this attribute. Only words with a (specifyable) minimum length are indexed. Word tokenisation is done using the word boundary analysis provided by libicu, which is taking into account the selected language provided at server start. Words are indexed in their lowercased form. The index supports complete match queries (full words) and prefix queries.

Geo Index: A geo index is used to find places on the surface of the earth fast.

Hash Index: A hash index is used to find documents based on examples.

Priority Queue: A priority queue based on an attribute of the documents.

Skiplist Index: A skiplist is used to find ranges of documents.

Key Generator: ArangoDB allows using key generators for each collection. Key generators have the purpose of auto-generating values for the `_key` attribute of a

document if none was specified by the user.

By default, ArangoDB will use the `traditional` key generator. The `traditional` key generator will auto-generate key values that are strings with ever-increasing numbers. The increment values it uses are non-deterministic.

Contrary, the `autoincrement` key generator will auto-generate deterministic key values. Both the start value and the increment value can be defined when the collection is created. The default start value is 0 and the default increment is 1, meaning the key values it will create by default are:

```
1, 2, 3, 4, 5, ...
```

When creating a collection with the `autoincrement` key generator and an `increment` of `5`, the generated keys would be:

```
1, 6, 11, 16, 21, ...
```