API Reference

Release 1.0.2

June 05, 2013

CONTENTS

CHAPTER

ONE

AUTHENTICATION METHODS

Most requests require the credentials of a Cloudant account. There are two ways to provide those credentials. They can either be provided using HTTP Basic Auth or as an HTTP cookie named AuthSession. The cookie can be obtained by performing a POST request to `/_session`. With the cookie set, information about the logged in user can be retrieved with a GET request and with a DELETE request you can end the session. Further details are provided below.

Method	Path	Description	Headers	Form Parameters
GET	/_session	Returns cookie based login user information	AuthSession cookie returned by POST request	—
POST	/_session	Do cookie based user login	`Content-Type: application/x-www-form-urlencoded`	username password
DELETE	/_session	Logout cookie based user	AuthSession cookie returned by POST request	—

Here is an example of a post request to obtain the authentication cookie.

```
POST /_session HTTP/1.1
Content-Length: 32
Content-Type: application/x-www-form-urlencoded
Accept: */*

name=YourUserName&password=YourPassword
```

And this is the corresponding reply with the Set-Cookie header.

```
200 OK
Cache-Control: must-revalidate
Content-Length: 42
Content-Type: text/plain; charset=UTF-8
Date: Mon, 04 Mar 2013 14:06:11 GMT
server: CouchDB/1.0.2 (Erlang OTP/R14B)
Set-Cookie: AuthSession="a2ltc3RlYmVsOjUxMzRBQTUzOtiY2_IDUIdsTJEVNEjObAbyhrgz"; Expires=Tue, 05 Ma
x-couch-request-id: a638431d

{
  "ok": true,
  "name": "kimstebel",
  "roles": []
}
```

Once you have obtained the cookie, you can make a GET request to obtain the username and its roles:

```
GET /_session HTTP/1.1
AuthSession: AuthSession="a2ltc3RlYmVsOjUxMzRBQTUzOtiY2_IDUIdsTJEVNEjObAbyhrgz"; Expires=Tue, 05 M
Accept: application/json
```

The body of the reply looks like this:

```
{
  "ok": true,
  "info": {
    "authentication_db": "_users",
    "authentication_handlers": ["cookie", "default"]
  },
  "userCtx": {
    "name": null,
    "roles": []
  }
}
```

To log out, you have to send a DELETE request to the same URL and sumbit the Cookie in the request.

```
DELETE /_session HTTP/1.1
AuthSession: AuthSession="a2ltc3RlYmVsOjUxMzRBQTUzOtiY2_IDUIdsTJEVNEjObAbyhrgz"; Expires=Tue, 05 M
Accept: application/json
```

This will result in the following response.

```
200 OK
Cache-Control: must-revalidate
Content-Length: 12
Content-Type: application/json
Date: Mon, 04 Mar 2013 14:06:12 GMT
server: CouchDB/1.0.2 (Erlang OTP/R14B)
Set-Cookie: AuthSession=""; Expires=Fri, 02 Jan 1970 00:00:00 GMT; Max-Age=0; Path=/; HttpOnly; Ve
x-couch-request-id: e02e0333

{
  "ok": true
}
```

CHAPTER

TWO

AUTHORIZATION SETTINGS

Cloudant's API allows you to read and modify the permissions of each user. Users are either identified by their Cloudant username or by their API key. You can also set permissions for unauthenticated users.

A list of the available methods and URL paths is provided below:

Method	Path	Description	Parameters
POST	https://cloudant.com/api/set_permissions	Set permissions for a user and database	database, username, roles[]
POST	https://cloudant.com/api/generate_api_key	Generate a random API key	—

2.1 Setting permissions

- **Method**: `POST https://cloudant.com/api/set_permissions`
- **Request Body**: Empty
- **Response**: JSON document indicating success or failure
- **Roles permitted**: admin
- **Query Arguments**:
  - **Argument**: database
    - **Description**: The database for which permissions are set. This has to be a string of the form "accountname/databasename".
    - **Optional**: no
    - **Type**: string
  - **Argument**: username
    - **Description**: The user name or API key for which permissions are set
    - **Optional**: yes
    - **Type**: string
  - **Argument**: roles
    - **Description**: The roles the user can have. This parameter can be passed multiple times for each role.
    - **Optional**: no
    - **Type**: string
    - **Supported Values**:
      - **_admin**: Gives the user all permissions, including setting permissions
      - **_reader**: Gives the user permission to read documents from the database

· **_writer**: Gives the user permission to create and modify documents in the database

- **Return Codes**:
  - **200**: Permissions have been set successfully.
  - **other values**: A Json document is returned describing the error. The document has the fields "error" and "reason".

2.1.1 Example Request

```
POST /api/set_permissions HTTP/1.1
Host: cloudant.com
Content-Length: 83
Content-Type: application/x-www-form-urlencoded

username=aUserNameOrApiKey&database=accountName/db&roles=_reader&roles=_writer
```

2.1.2 Example Response

```
{
  "ok": true
}
```

2.2 Generating an API Key

- **Method**: `POST https://cloudant.com/api/generate_api_key`
- **Request Body**: Empty
- **Response**: JSON document containing the generated key and password
- **Roles permitted**: admin
- **Query Arguments**: none
- **Return Codes**:
  - **200**: Permissions have been set successfully.
  - **other values**: A Json document is returned describing the error. The document has the fields "error" and "reason".

2.2.1 Structure of the JSON document returned

- **ok**: true if request was successful
- **key**: String containing the generated key
- **password**: String containing the generated password

2.2.2 Example Request

```
POST /api/generate_api_key HTTP/1.1
Host: cloudant.com
```

2.2.3 Example Response

```
{
  "password": "generatedPassword",
   "ok": true,
   "key": "generatedKey"
}
```

CHAPTER

THREE

DATABASE METHODS

The Database methods provide an interface to an entire database within Cloudant. These are database level rather than document level requests.

A list of the available methods and URL paths are provided below:

Method	Path	Description
GET	/_all_dbs	Returns a list of all databases
GET	/db	Returns database information
PUT	/db	Create a new database
DELETE	/db	Delete an existing database
GET	/db/_all_docs	Returns a built-in view of all documents in this database
POST	/db/_all_docs	Returns certain rows from the built-in view of all documents
POST	/db/_bulk_docs	Insert multiple documents in to the database in a single request
GET	/db/_changes	Returns changes for the given database
POST	/db/_ensure_full_commit	Makes sure all uncommitted changes are written and synchronized to the disk
POST	/db/_missing_revs	Given a list of document revisions, returns the document revisions that do not exist in the database
POST	/db/_revs_diff	Given a list of document revisions, returns differences between the given revisions and ones that are in the database
GET	/db/_revs_limit	Gets the limit of historical revisions to store for a single document in the database
PUT	/db/_revs_limit	Sets the limit of historical revisions to store for a single document in the database
GET	/db/_security	Returns the special security object for the database
PUT	/db/_security	Sets the special security object for the database
POST	/db/_view_cleanup	Removes view files that are not used by any design document

3.1 Retrieving a list of all databases

- **Method**: `GET /_all_dbs`
- **Request**: None
- **Response**: JSON list of DBs
- **Roles permitted**: _reader
- **Return Codes**:
  - **200**: Request completed successfully.

Returns a list of all the databases. For example:

```
GET http://username.cloudant.com/_all_dbs
Accept: application/json
```

The return is a JSON array:

```
[
   "_users",
   "contacts",
   "docs",
   "invoices",
   "locations"
]
```

3.2 Operations on entire databases

For all the database methods, the database name within the URL path should be the database name that you wish to perform the operation on. For example, to obtain the meta information for the database `recipes`, you would use the HTTP request:

```
GET /recipes
```

For clarity, the form below is used in the URL paths:

```
GET /db
```

Where `db` is the name of any database.

3.2.1 Retrieving information about a database

- **Method**: `GET /db`
- **Request**: None
- **Response**: Information about the database in JSON format
- **roles permitted**: _reader, _admin
- **Return Codes**:
  - **404**: The requested content could not be found. If further information is available, it will be returned as a JSON object.

Gets information about the specified database. For example, to retrieve the information for the database `recipe`:

```
GET /db HTTP/1.1
Accept: application/json
```

The JSON response contains meta information about the database. A sample of the JSON returned for an empty database is provided below:

```
{
  "update_seq": "0-g1AAAADneJzLYWBgYMlgTmFQSElKzi9KdUhJMtbLTS3KLElMT9VLzskvTUnMK9HLSy3JAapkSmRIsv
  "db_name": "db",
  "purge_seq": 0,
  "other": {
    "data_size": 0
  },
  "doc_del_count": 0,
  "doc_count": 0,
  "disk_size": 316,
  "disk_format_version": 5,
  "compact_running": false,
  "instance_start_time": "0"
}
```

The elements of the returned structure are shown in the table below:

Field	Description
compact_running	Set to true if the database compaction routine is operating on this database.
db_name	The name of the database.
disk_format_version	The version of the physical format used for the data when it is stored on disk.
disk_size	Size in bytes of the data as stored on the disk. Views indexes are not included in the calculation.
doc_count	A count of the documents in the specified database.
doc_del_count	Number of deleted documents
instance_start_time	Always 0.
purge_seq	The number of purge operations on the database.
update_seq	The current number of updates to the database.
other	Json object containing a data_size field.

3.2.2 Creating a database

- **Method**: `PUT /db`
- **Request**: None
- **Response**: JSON success statement
- **roles permitted**: _admin
- **Return Codes**:
  - **403**: Invalid database name
  - **412**: Database already exists

Creates a new database. The database name must be composed of one or more of the following characters:

- Lowercase characters (`a-z`)
- Name must begin with a lowercase letter
- Digits (`0-9`)
- Any of the characters _, `$`, `(`, `)`, `+`, `-`, and `/`.

Trying to create a database that does not meet these requirements will return an error quoting these restrictions.

To create the database `recipes`:

```
PUT /db HTTP/1.1
Accept: application/json
```

The returned content contains the JSON status:

```
{
  "ok": true
}
```

Anything else should be treated as an error, and the problem should be taken from the HTTP response code.

3.2.3 Deleting a database

- **Method**: `DELETE /db`
- **Request**: None
- **Response**: JSON success statement
- **roles permitted**: _admin

- **Return Codes**:
  - **200**: Database has been deleted
  - **404**: The requested content could not be found. If further information is available, it will be returned as a JSON object.

Deletes the specified database, and all the documents and attachments contained within it.

To delete the database `recipes` you would send the request:

```
DELETE /db HTTP/1.1
Accept: application/json
```

If successful, the returned JSON will indicate success

```
{
  "ok": true
}
```

3.3 Retrieving multiple documents in one request

3.3.1 GET /db/_all_docs

- **Method**: `GET /db/_all_docs`
- **Request**: None
- **Response**: JSON object containing document information, ordered by the document ID
- **roles permitted**: _admin, _reader
- **Query Arguments**:
  - **Argument**: descending
    - **Description**: Return the documents in descending by key order
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: false
  - **Argument**: endkey
    - **Description**: Stop returning records when the specified key is reached
    - **Optional**: yes
    - **Type**: string
  - **Argument**: endkey_docid
    - **Description**: Stop returning records when the specified document ID is reached
    - **Optional**: yes
    - **Type**: string
  - **Argument**: group
    - **Description**: Group the results using the reduce function to a group or single row
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: false
  - **Argument**: group_level

        * **Description**: Specify the group level to be used
        * **Optional**: yes
        * **Type**: numeric
    - **Argument**: include_docs
        * **Description**: Include the full content of the documents in the return
        * **Optional**: yes
        * **Type**: boolean
        * **Default**: false
    - **Argument**: inclusive_end
        * **Description**: Specifies whether the specified end key should be included in the result
        * **Optional**: yes
        * **Type**: boolean
        * **Default**: true
    - **Argument**: key
        * **Description**: Return only documents that match the specified key
        * **Optional**: yes
        * **Type**: string
    - **Argument**: limit
        * **Description**: Limit the number of the returned documents to the specified number
        * **Optional**: yes
        * **Type**: numeric
    - **Argument**: reduce
        * **Description**: Use the reduction function
        * **Optional**: yes
        * **Type**: boolean
        * **Default**: true
    - **Argument**: skip
        * **Description**: Skip this number of records before starting to return the results
        * **Optional**: yes
        * **Type**: numeric
        * **Default**: 0
    - **Argument**: stale
        * **Description**: Allow the results from a stale view to be used
        * **Optional**: yes
        * **Type**: string
        * **Default**:
        * **Supported Values**:
            - **ok**: Allow stale views
    - **Argument**: startkey

        * **Description**: Return records starting with the specified key
        * **Optional**: yes
        * **Type**: string
    - **Argument**: startkey_docid
        * **Description**: Return records starting with the specified document ID
        * **Optional**: yes
        * **Type**: string

Returns a JSON structure of all of the documents in a given database. The information is returned as a JSON structure containing meta information about the return structure, and the list documents and basic contents, consisting the ID, revision and key. The key is generated from the document ID.

Field	Description
offset	Offset where the document list started
rows [array]	Array of document object
total_rows	Number of documents in the database/view
update_seq	Current update sequence for the database

By default the information returned contains only the document ID and revision. For example, the request:

```
GET /test/_all_docs HTTP/1.1
Accept: application/json
```

Returns the following structure:

```
{
  "total_rows": 3,
  "offset": 0,
  "rows": [{
    "id": "5a049246-179f-42ad-87ac-8f080426c17c",
    "key": "5a049246-179f-42ad-87ac-8f080426c17c",
    "value": {
      "rev": "2-9d5401898196997853b5ac4163857a29"
    }
  }, {
    "id": "96f898f0-f6ff-4a9b-aac4-503992f31b01",
    "key": "96f898f0-f6ff-4a9b-aac4-503992f31b01",
    "value": {
      "rev": "2-ff7b85665c4c297838963c80ecf481a3"
    }
  }, {
    "id": "d1f61e66-7708-4da6-aa05-7cbc33b44b7e",
    "key": "d1f61e66-7708-4da6-aa05-7cbc33b44b7e",
    "value": {
      "rev": "2-cbdef49ef3ddc127eff86350844a6108"
    }
  }]
}
```

The information is returned in the form of a temporary view of all the database documents, with the returned key consisting of the ID of the document. The remainder of the interface is therefore identical to the View query arguments and their behavior.

3.3.2 `POST /db/_all_docs`

- **Method**: `POST /db/_all_docs`
- **Request**: JSON of the document IDs you want included
- **Response**: JSON of the returned view

- **roles permitted**: _admin, _reader

The `POST` to `_all_docs` allows to specify multiple keys to be selected from the database. This enables you to request multiple documents in a single request, in place of multiple *Retrieving a document* requests.

The request body should contain a list of the keys to be returned as an array to a `keys` object. For example:

```
POST /recipes/_all_docs
User-Agent: MyApp/0.1 libwww-perl/5.837

{
   "keys" : [
      "Zingylemontart",
      "Yogurtraita"
   ]
}
```

The return JSON is the all documents structure, but with only the selected keys in the output:

```
{
   "total_rows" : 2666,
   "rows" : [
      {
         "value" : {
            "rev" : "1-a3544d296de19e6f5b932ea77d886942"
         },
         "id" : "Zingylemontart",
         "key" : "Zingylemontart"
      },
      {
         "value" : {
            "rev" : "1-91635098bfe7d40197a1b98d7ee085fc"
         },
         "id" : "Yogurtraita",
         "key" : "Yogurtraita"
      }
   ],
   "offset" : 0
}
```

3.4 Creating or updating multiple documents

- **Method**: `POST /db/_bulk_docs`
- **Request**: JSON of the docs and updates to be applied
- **Response**: JSON success statement
- **roles permitted**: _admin, _writer
- **Return Codes**:
  - **201**: Document(s) have been created or updated

The bulk document API allows you to create and update multiple documents at the same time within a single request. The basic operation is similar to creating or updating a single document, except that you batch the document structure and information and . When creating new documents the document ID is optional. For updating existing documents, you must provide the document ID, revision information, and new document values.

For both inserts and updates the basic structure of the JSON is the same:

Field	Description
docs [array]	Bulk Documents Document
_id (optional)	List of changes, field-by-field, for this document
_rev (optional)	Document ID
_deleted (optional)	Update sequence number

3.4.1 Inserting Documents in Bulk

To insert documents in bulk into a database you need to supply a JSON structure with the array of documents that you want to add to the database. Using this method you can either include a document ID, or allow the document ID to be automatically generated.

For example, the following inserts three new documents with the supplied document IDs. If you omit the document ID, it will be generated:

```
{
  "docs": [{
    "name": "Nicholas",
    "age": 45,
    "gender": "male",
    "_id": "96f898f0-f6ff-4a9b-aac4-503992f31b01",
    "_attachments": {

    }
  }, {
    "name": "Taylor",
    "age": 50,
    "gender": "male",
    "_id": "5a049246-179f-42ad-87ac-8f080426c17c",
    "_attachments": {

    }
  }, {
    "name": "Owen",
    "age": 51,
    "gender": "male",
    "_id": "d1f61e66-7708-4da6-aa05-7cbc33b44b7e",
    "_attachments": {

    }
  }]
}
```

The return type from a bulk insertion will be 201, with the content of the returned structure indicating specific success or otherwise messages on a per-document basis.

The return structure from the example above contains a list of the documents created, here with the combination and their revision IDs:

```
201 Created
Cache-Control: must-revalidate
Content-Length: 269
Content-Type: application/json
Date: Mon, 04 Mar 2013 14:06:20 GMT
server: CouchDB/1.0.2 (Erlang OTP/R14B)
x-couch-request-id: e8ff64d5

[{
  "id": "96f898f0-f6ff-4a9b-aac4-503992f31b01",
  "rev": "1-54dd23d6a630d0d75c2c5d4ef894454e"
}, {
  "id": "5a049246-179f-42ad-87ac-8f080426c17c",
```

```
  "rev": "1-0cde94a828df5cdc0943a10f3f36e7e5"
}, {
  "id": "d1f61e66-7708-4da6-aa05-7cbc33b44b7e",
  "rev": "1-a2b6e5dac4e0447e7049c8c540b309d6"
}]
```

The content and structure of the returned JSON will depend on the transaction semantics being used for the bulk update; see *Bulk Documents Transaction Semantics* for more information. Conflicts and validation errors when updating documents in bulk must be handled separately; see *Bulk Document Validation and Conflict Errors*.

Updating Documents in Bulk

The bulk document update procedure is similar to the insertion procedure, except that you must specify the document ID and current revision for every document in the bulk update JSON string.

For example, you could send the following request:

```
POST /test/_bulk_docs HTTP/1.1
Accept: application/json

{
  "docs": [{
    "name": "Nicholas",
    "age": 45,
    "gender": "female",
    "_id": "96f898f0-f6ff-4a9b-aac4-503992f31b01",
    "_attachments": {

    },
    "_rev": "1-54dd23d6a630d0d75c2c5d4ef894454e"
  }, {
    "name": "Taylor",
    "age": 50,
    "gender": "female",
    "_id": "5a049246-179f-42ad-87ac-8f080426c17c",
    "_attachments": {

    },
    "_rev": "1-0cde94a828df5cdc0943a10f3f36e7e5"
  }, {
    "name": "Owen",
    "age": 51,
    "gender": "female",
    "_id": "d1f61e66-7708-4da6-aa05-7cbc33b44b7e",
    "_attachments": {

    },
    "_rev": "1-a2b6e5dac4e0447e7049c8c540b309d6"
  }]
}
```

The return structure is the JSON of the updated documents, with the new revision and ID information:

```
[{
  "id": "96f898f0-f6ff-4a9b-aac4-503992f31b01",
  "rev": "2-ff7b85665c4c297838963c80ecf481a3"
}, {
  "id": "5a049246-179f-42ad-87ac-8f080426c17c",
  "rev": "2-9d5401898196997853b5ac4163857a29"
}, {
  "id": "d1f61e66-7708-4da6-aa05-7cbc33b44b7e",
```

```
  "rev": "2-cbdef49ef3ddc127eff86350844a6108"
}]
```

You can optionally delete documents during a bulk update by adding the `_deleted` field with a value of `true` to each document ID/revision combination within the submitted JSON structure.

The return type from a bulk insertion will be 201, with the content of the returned structure indicating specific success or otherwise messages on a per-document basis.

The content and structure of the returned JSON will depend on the transaction semantics being used for the bulk update; see *Bulk Documents Transaction Semantics* for more information. Conflicts and validation errors when updating documents in bulk must be handled separately; see *Bulk Document Validation and Conflict Errors*.

Bulk Documents Transaction Semantics

Cloudant will only guarantee that some of the documents will be saved when you send the request. The response will contain the list of documents successfully inserted or updated during the process. In the event of a crash, some of the documents may have been successfully saved, and some will have been lost.

The response structure will indicate whether the document was updated by supplying the new `_rev` parameter indicating a new document revision was created. If the update failed, then you will get an `error` of type `conflict`. For example:

```
[
   {
      "id" : "FishStew",
      "error" : "conflict",
      "reason" : "Document update conflict."
   },
   {
      "id" : "LambStew",
      "error" : "conflict",
      "reason" : "Document update conflict."
   },
   {
      "id" : "7f7638c86173eb440b8890839ff35433",
      "error" : "conflict",
      "reason" : "Document update conflict."
   }
]
```

In this case no new revision has been created and you will need to submit the document update with the correct revision tag, to update the document.

Bulk Document Validation and Conflict Errors

The JSON returned by the `_bulk_docs` operation consists of an array of JSON structures, one for each document in the original submission. The returned JSON structure should be examined to ensure that all of the documents submitted in the original request were successfully added to the database.

The exact structure of the returned information is:

Field	Description
docs [array]	Bulk Documents Document
id	Document ID
error	Error type
reason	Error string with extended reason

When a document (or document revision) is not correctly committed to the database because of an error, you should check the `error` field to determine error type and course of action. Errors will be one of the following type:

- `conflict`

  The document as submitted is in conflict. If you used the default bulk transaction mode then the new revision will not have been created and you will need to re-submit the document to the database.

  Conflict resolution of documents added using the bulk docs interface is identical to the resolution procedures used when resolving conflict errors during replication.

- `forbidden`

  Entries with this error type indicate that the validation routine applied to the document during submission has returned an error.

  For example, if your validation routine includes the following:

  ```
  throw({forbidden: 'invalid recipe ingredient'});
  ```

  The error returned will be:

  ```
  {
     "id" : "7f7638c86173eb440b8890839ff35433",
     "error" : "forbidden",
     "reason" : "invalid recipe ingredient"
  }
  ```

3.5 Obtaining a list of changes

- **Method**: `GET /db/_changes`
- **Request**: None
- **Response**: JSON success statement
- **Roles permitted**: _admin, _reader
- **Query Arguments**:
  - **Argument**: doc_ids
    - **Description**: Specify the list of documents IDs to be filtered
    - **Optional**: yes
    - **Type**: json
    - **Default**: none
  - **Argument**: feed
    - **Description**: Type of feed
    - **Optional**: yes
    - **Type**: string
    - **Default**: normal
    - **Supported Values**:
      - **continuous**: Continuous (non-polling) mode
      - **longpoll**: Long polling mode
      - **normal**: Normal mode
  - **Argument**: filter
    - **Description**: Filter function from a design document to get updates
    - **Optional**: yes

        * **Type**: string
        * **Default**: none
        * **Supported Values**:
    - **Argument**: heartbeat
        * **Description**: Period after which an empty line is sent during longpoll or continuous
        * **Optional**: yes
        * **Type**: numeric
        * **Default**: 60000
        * **Quantity**: milliseconds
    - **Argument**: include_docs
        * **Description**: Include the document with the result
        * **Optional**: yes
        * **Type**: boolean
        * **Default**: false
    - **Argument**: limit
        * **Description**: Maximum number of rows rows to return
        * **Optional**: yes
        * **Type**: numeric
        * **Default**: none
    - **Argument**: since
        * **Description**: Start the results from changes immediately after the specified sequence number
        * **Optional**: yes
        * **Type**: numeric
        * **Default**: 0
    - **Argument**: descending
        * **Description**: Return the changes in descending (by `seq`) order
        * **Optional**: yes
        * **Type**: boolean
        * **Default**: false

Obtains a list of the changes made to the database. This can be used to monitor for update and modifications to the database for post processing or synchronization. The _changes feed is not guaranteed to return changes in the correct order. There are three different types of supported changes feeds, poll, longpoll, and continuous. All requests are poll requests by default. You can select any feed type explicitly using the `feed` query argument.

- **Poll**

  With polling you can request the changes that have occured since a specific sequence number. This returns the JSON structure containing the changed document information. When you perform a poll change request, only the changes since the specific sequence number are returned. For example, the query

  ```
  GET /recipes/_changes
  Content-Type: application/json
  ```

Will get all of the changes in the database. You can request a starting point using the `since` query argument and specifying the sequence number. You will need to record the latest sequence number in your client and then use this when making another request as the new value to the `since` parameter.

- **Longpoll**

  With long polling the request to the server will remain open until a change is made on the database, when the changes will be reported, and then the connection will close. The long poll is useful when you want to monitor for changes for a specific purpose without wanting to monitoring continuously for changes.

  Because the wait for a change can be significant you can set a timeout before the connection is automatically closed (the `timeout` argument). You can also set a heartbeat interval (using the `heartbeat` query argument), which sends a newline to keep the connection open.

- **Continuous**

  Continuous sends all new changes back to the client immediately, without closing the connection. In continuous mode the format of the changes is slightly different to accommodate the continuous nature while ensuring that the JSON output is still valid for each change notification.

  As with the longpoll feed type you can set both the timeout and heartbeat intervals to ensure that the connection is kept open for new changes and updates.

The return structure for `normal` and `longpoll` modes is a JSON array of changes objects, and the last update sequence number. The structure is described in the following table.

Field	Description
last_seq	Last change sequence number.
results [array]	Changes made to a database
changes [array]	List of changes, field-by-field, for this document
id	Document ID
seq	Update sequence number

The return format for `continuous` mode the server sends a `CRLF` (carriage-return, linefeed) delimited line for each change. Each line contains the JSON object.

You can also request the full contents of each document change (instead of just the change notification) by using the `include_docs` parameter.

3.5.1 Filtering

You can filter the contents of the changes feed in a number of ways. The most basic way is to specify one or more document IDs to the query. This causes the returned structure value to only contain changes for the specified IDs. Note that the value of this query argument should be a JSON formatted array.

You can also filter the `_changes` feed by defining a filter function within a design document. The specification for the filter is the same as for replication filters. You specify the name of the filter function to the `filter` parameter, specifying the design document name and filter name. For example:

```
GET /db/_changes?filter=design_doc/filtername
```

The `_changes` feed can be used to watch changes to specific document ID's or the list of `_design` documents in a database. If the `filters` parameter is set to `_doc_ids` a list of doc IDs can be passed in the `doc_ids` parameter as a JSON array.

3.6 Cleaning up cached view output

- **Method**: `POST /db/_view_cleanup`
- **Request**: None
- **Response**: JSON success statement

- **Roles permitted**: _admin

Cleans up the cached view output on disk for a given view. For example:

```
POST /recipes/_view_cleanup
Content-Type: application/json
```

If the request is successful, a basic status message us returned:

```
{
   "ok" : true
}
```

3.7 Commiting all changes to disk

- **Method**: `POST /db/_ensure_full_commit`
- **Request**: None
- **Response**: JSON success statement
- **Roles permitted**: _admin, _writer
- **Return Codes**:
  - **202**: Commit completed successfully
  - **404**: The requested content could not be found. The returned content will include further information, as a JSON object, if available.

Commits any recent changes to the specified database to disk. You should call this if you want to ensure that recent changes have been written. For example, to commit all the changes to disk for the database `recipes` you would use:

```
POST /recipes/_ensure_full_commit
Content-Type: application/json
```

This returns a status message containing the success message and a timestamp.

```
{
  "ok" : true,
  "instance_start_time" : "0"
}
```

3.8 Retrieving missing revisions

- **Method**: `POST /db/_missing_revs`
- **Request**: JSON list of document revisions
- **Response**: JSON of missing revisions

3.9 Retrieving differences between revisions

- **Method**: `POST /db/_revs_diff`
- **Request**: JSON list of document revisions
- **Response**: JSON list of differences from supplied document/revision list

3.10 The database security document

3.10.1 Retrieving the security document

- **Method**: `GET /db/_security`
- **Request**: None
- **Response**: JSON of the security object

Gets the current security object from the specified database. The security object consists of two compulsory elements, `admins` and `readers`, which are used to specify the list of users and/or roles that have admin and reader rights to the database respectively. Any additional fields in the security object are optional. The entire security object is made available to validation and other internal functions so that the database can control and limit functionality.

To get the existing security object you would send the following request:

```
{
   "admins" : {
      "roles" : [],
      "names" : [
         "mc",
         "slp"
      ]
   },
   "readers" : {
      "roles" : [],
      "names" : [
         "tim",
         "brian"
      ]
   }
}
```

Security object structure is:

- **admins**: Roles/Users with admin privileges
  - **roles** [array]: List of roles with parent privilege
  - **users** [array]: List of users with parent privilege
- **readers**: Roles/Users with reader privileges
  - **roles** [array]: List of roles with parent privilege
  - **users** [array]: List of users with parent privilege

Note: If the security object for a database has never been set, then the value returned will be empty.

3.10.2 Creating or updating the security document

- **Method**: `PUT /db/_security`
- **Request**: JSON specifying the admin and user security for the database
- **Response**: JSON status message

Sets the security object for the given database. For example, to set the security object for the `recipes` database:

```
PUT http://username.cloudant.com/recipes/_security
Content-Type: application/json
```

```
{
   "admins" : {
      "roles" : [],
      "names" : [
         "mc",
         "slp"
      ]
   },
   "readers" : {
      "roles" : [],
      "names" : [
         "tim",
         "brian"
      ]
   }
}
```

If the setting was successful, a JSON status object will be returned:

```
{
   "ok" : true
}
```

3.11 The revisions limit

3.11.1 Retrieving the revisions limit

- **Method**: `GET /db/_revs_limit`
- **Request**: None
- **Response**: The current revision limit setting
- **Roles permitted**: _admin, _reader

Gets the current `revs_limit` (revision limit) setting.

For example to get the current limit:

```
GET /recipes/_revs_limit
Content-Type: application/json
```

The returned information is the current setting as a numerical scalar:

```
1000
```

3.11.2 Setting the revisions limit

- **Method**: `PUT /db/_revs_limit`
- **Request**: A scalar integer of the revision limit setting
- **Response**: Confirmation of setting of the revision limit
- **Roles permitted**: _admin, _writer

Sets the maximum number of document revisions that will be tracked even after compaction has occurred. You can set the revision limit on a database by using `PUT` with a scalar integer of the limit that you want to set as the request body.

For example to set the revs limit to 100 for the `recipes` database:

```
PUT /recipes/_revs_limit
Content-Type: application/json

100
```

If the setting was successful, a JSON status object will be returned:

```
{
   "ok" : true
}
```

CHAPTER

FOUR

DOCUMENT METHODS

The document methods can be used to create, read, update and delete documents within a database.

A list of the available methods and URL paths is provided below:

Method	Path	Description
POST	/db	Create a new document
GET	/db/doc	Returns the latest revision of the document
HEAD	/db/doc	Returns bare information in the HTTP Headers for the document
PUT	/db/doc	Inserts a new document, or new version of an existing document
DELETE	/db/doc	Deletes the document
COPY	/db/doc	Copies the document
GET	/db/doc/attachment	Gets the attachment of a document
PUT	/db/doc/attachment	Adds an attachment of a document
DELETE	/db/doc/attachment	Deletes an attachment of a document

4.1 CRUD operations on documents

4.1.1 Creating a new document

- **Method**: `POST /db`
- **Request**: JSON of the new document
- **Response**: JSON with the committed document information
- **Roles permitted**: _writer
- **Query Arguments**:
  - **Argument**: batch
    - **Description**: Allow document store request to be batched with others
    - **Optional**: yes
    - **Type**: string
    - **Supported Values**: asd
    - **ok**: Enable
- **Return Codes**:
  - **201**: Document has been created successfully
  - **409**: Conflict - a document with the specified document ID already exists

Create a new document in the specified database, using the supplied JSON document structure. If the JSON structure includes the `_id` field, then the document will be created with the specified document ID. If the `_id` field is not specified, a new unique ID will be generated.

For example, you can generate a new document with a generated UUID using the following request:

```
POST /recipes/
Content-Type: application/json

{
   "servings" : 4,
   "subtitle" : "Delicious with fresh bread",
   "title" : "Fish Stew"
}
```

The return JSON will specify the automatically generated ID and revision information:

```
{
   "id" : "64575eef70ab90a2b8d55fc09e00440d",
   "ok" : true,
   "rev" : "1-9c65296036141e575d32ba9c034dd3ee"
}
```

Specifying the Document ID

The document ID can be specified by including the `_id` field in the JSON of the submitted record. The following request will create the same document with the ID `FishStew`:

```
POST /recipes/
Content-Type: application/json

{
   "_id" : "FishStew",
   "servings" : 4,
   "subtitle" : "Delicious with fresh bread",
   "title" : "Fish Stew"
}
```

The structure of the submitted document is as shown in the table below:

In either case, the returned JSON will specify the document ID, revision ID, and status message:

```
{
   "id" : "FishStew",
   "ok" : true,
   "rev" : "1-9c65296036141e575d32ba9c034dd3ee"
}
```

UUID generation algorithms

A number of different UUID generation algorithms are provided. You can use them in situations where a user-specified UUID does not make sense. These can be set by *PUT /_config/uuids/algorithm*.

Algorithm	Description	Sample UUID
random	128 bits of pure random awesomeness	43febce5675468a5467fb5467ce9e6c0
sequential	monotonically increasing ids with random increments	f755c413badf66b22941313f9f001e28 f755c413badf66b22941313f9f0024ca f755c413badf66b22941313f9f00332c
utc_random	time since start of epoch, as 14 hex digits, followed by 18 random digits.	04cfa405381205204f75100d0241ccc3 04cfa4059c48e76e7c054bbe033dd8db 04cfa405fce10b0df4c08f95e667cd2f
utc_id & additional parameter	time since start of epoch, as 14 hex digits, followed by utc_id_suffix.	04cfa718b00848_i_am_in_yer_couch 04cfa71d377aef_i_am_in_yer_couch 04cfa71e0deabd_i_am_in_yer_couch

Batch Mode Writes

You can write documents to the database at a higher rate by using the batch option. This collects document writes together in memory (on a user-by-user basis) before they are committed to disk. This increases the risk of the documents not being stored in the event of a failure, since the documents are not written to disk immediately.

To use the batched mode, append the `batch=ok` query argument to the URL of the `PUT` or `POST` request. The server will respond with a 202 HTTP response code immediately.

Including Attachments

You can include one or more attachments with a given document by incorporating the attachment information within the JSON of the document. This provides a simpler alternative to loading documents with attachments than making a separate call (see *Creating or updating an attachment*).

- **_id** (optional): Document ID
- **_rev** (optional): Revision ID (when updating an existing document)
- **_attachments** (optional): Document Attachment
  - **filename**: Attachment information
    - **content_type**: MIME Content type string
    - **data**: File attachment content, Base64 encoded

The `filename` will be the attachment name. For example, when sending the JSON structure below:

```
{
   "_id" : "FishStew",
   "servings" : 4,
   "subtitle" : "Delicious with fresh bread",
   "title" : "Fish Stew"
   "_attachments" : {
      "styling.css" : {
         "content-type" : "text/css",
         "data" : "cCB7IGZvbnQtc2l6ZTogMTJwdDsgfQo=",
         },
   },
}
```

The attachment `styling.css` can be accessed using `/recipes/FishStew/styling.css`. For more information on attachments, see *Attachments*.

The document data embedded into the structure must be encoded using base64.

4.1.2 Retrieving a document

- **Method**: `GET /db/doc`
- **Request**: None
- **Response**: Returns the JSON for the document
- **Roles permitted**: _reader
- **Query Arguments**:
  - **Argument**: conflicts
    - **Description**: Returns the conflict tree for the document.
    - **Optional**: yes
    - **Type**: boolean

        - **Default**: false
        - **Supported Values**:
            - **true**: Includes conflicting revisions
    - **Argument**: rev
        - **Description**: Specify the revision to return
        - **Optional**: yes
        - **Type**: string
        - **Supported Values**:
            - **true**: Includes the revisions
    - **Argument**: revs
        - **Description**: Return a list of the revisions for the document
        - **Optional**: yes
        - **Type**: boolean
    - **Argument**: revs_info
        - **Description**: Return a list of detailed revision information for the document
        - **Optional**: yes
        - **Type**: boolean
        - **Supported Values**:
            - **true**: Includes the revisions
- **Return Codes**:
    - **200**: Document retrieved
    - **400**: The format of the request or revision was invalid
    - **404**: The specified document or revision cannot be found, or has been deleted
    - **409**: Conflict - a document with the specified document ID already exists

Returns the specified `doc` from the specified `db`. For example, to retrieve the document with the id `DocID` you would send the following request:

```
GET /db/DocID HTTP/1.1
Accept: application/json
```

The returned JSON is the JSON of the document, including the document ID and revision number:

```
{
  "_id": "DocID",
  "_rev": "1-2b458b0705e3007bce80b0499a1199e7",
  "name": "Anna",
  "age": 89,
  "gender": "female"
}
```

Unless you request a specific revision, the latest revision of the document will always be returned.

Attachments

If the document includes attachments, then the returned structure will contain a summary of the attachments associated with the document, but not the attachment data itself.

The JSON for the returned document will include the `_attachments` field, with one or more attachment definitions. For example:

```
{
  "_id": "DocID",
  "_rev": "2-f29c836d0bedc4b4b95cfaa6d99e95df",
  "name": "Anna",
  "age": 89,
  "gender": "female",
  "_attachments": {
    "my attachment": {
      "content_type": "application/json; charset=UTF-8",
      "revpos": 2,
      "digest": "md5-37IZysiyWLRWx31J/1WQHw==",
      "length": 12,
      "stub": true
    }
  }
}
```

The format of the returned JSON is shown in the table below:

- **_id** (optional): Document ID
- **_rev** (optional): Revision ID (when updating an existing document)
- **_attachments** (optional): Document Attachment
  - **filename**: Attachment information
    - **content_type**: MIME Content type string
    - **length**: Length (bytes) of the attachment data
    - **revpos**: Revision where this attachment exists
    - **digest**: MD5 checksum of the attachment
    - **stub**: Indicates whether the attachment is a stub

Getting a List of Revisions

You can obtain a list of the revisions for a given document by adding the `revs=true` parameter to the request URL. For example:

```
GET /recipes/FishStew?revs=true
Accept: application/json
```

The returned JSON structure includes the original document, including a `_revisions` structure that includes the revision information:

```
{
   "servings" : 4,
   "subtitle" : "Delicious with a green salad",
   "_id" : "FishStew",
   "title" : "Irish Fish Stew",
   "_revisions" : {
      "ids" : [
         "a1a9b39ee3cc39181b796a69cb48521c",
         "7c4740b4dcf26683e941d6641c00c39d",
         "9c65296036141e575d32ba9c034dd3ee"
      ],
      "start" : 3
   },
   "_rev" : "3-a1a9b39ee3cc39181b796a69cb48521c"
}
```

- **_id** (optional): Document ID
- **_rev** (optional): Revision ID (when updating an existing document)
- **_revisions**: Document Revisions
  - **ids** [array]: Array of valid revision IDs, in reverse order (latest first)
  - **start**: Prefix number for the latest revision

Obtaining an Extended Revision History

You can get additional information about the revisions for a given document by supplying the `revs_info` argument to the query:

```
GET /recipes/FishStew?revs_info=true
Accept: application/json
```

This returns extended revision information, including the availability and status of each revision:

```
{
   "servings" : 4,
   "subtitle" : "Delicious with a green salad",
   "_id" : "FishStew",
   "_revs_info" : [
      {
         "status" : "available",
         "rev" : "3-a1a9b39ee3cc39181b796a69cb48521c"
      },
      {
         "status" : "available",
         "rev" : "2-7c4740b4dcf26683e941d6641c00c39d"
      },
      {
         "status" : "available",
         "rev" : "1-9c65296036141e575d32ba9c034dd3ee"
      }
   ],
   "title" : "Irish Fish Stew",
   "_rev" : "3-a1a9b39ee3cc39181b796a69cb48521c"
}
```

- **_id** (optional): Document ID
- **_rev** (optional): Revision ID (when updating an existing document)
- **_revs_info** [array]: Document Extended Revision Info
  - **rev**: Full revision string
  - **status**: Status of the revision

Obtaining a Specific Revision

To get a specific revision, use the `rev` argument to the request, and specify the full revision number:

```
GET /recipes/FishStew?rev=2-7c4740b4dcf26683e941d6641c00c39d
Accept: application/json
```

The specified revision of the document will be returned, including a `_rev` field specifying the revision that was requested:

```
{
   "_id" : "FishStew",
   "_rev" : "2-7c4740b4dcf26683e941d6641c00c39d",
   "servings" : 4,
   "subtitle" : "Delicious with a green salad",
   "title" : "Fish Stew"
}
```

Retrieving conflicting revisions

To get a list of conflicting revisions, set the `conflicts` argument to `true`.

```
GET /recipes/FishStew?conflicts=true
Accept: application/json
```

If there are conflicts, the returned document will include a `_conflicts` field specifying the revisions that are in conflict.

```
{
   "_id" : "FishStew",
   "_rev" : "2-7c4740b4dcf26683e941d6641c00c39d",
   "servings" : 4,
   "subtitle" : "Delicious with a green salad",
   "title" : "Fish Stew",
   "_conflicts": ["2-65db2a11b5172bf928e3bcf59f728970","2-5bc3c6319edf62d4c624277fdd0ae191"]
}
```

Overriding the default read quorum

As in the case of updates there is an r query-string parameter that sets the quorum for reads. When a document is read, requests are issued to all N copies of the partition hosting the document and the client receives a response when r matching success responses are received. The default quorum is the simple majority of N, which is the recommended choice for most applications.

4.1.3 Retrieving revision and size of a document

- **Method**: `HEAD /db/doc`
- **Request**: None
- **Response**: None
- **Roles permitted**: _reader
- **Query Arguments**:
  - **Argument**: rev
    - **Description**: Specify the revision to return
    - **Optional**: yes
    - **Type**: string
  - **Argument**: revs
    - **Description**: Return a list of the revisions for the document
    - **Optional**: yes
    - **Type**: boolean
  - **Argument**: revs_info

        - **Description**: Return a list of detailed revision information for the document
        - **Optional**: yes
        - **Type**: boolean

- **Return Codes**:
  - **404**: The specified document or revision cannot be found, or has been deleted

Returns the HTTP Headers containing a minimal amount of information about the specified document. The method supports the same query arguments as the `GET` method, but only the header information (including document size, and the revision as an ETag), is returned. For example, a simple `HEAD` request:

```
HEAD /recipes/FishStew
Content-Type: application/json
```

Returns the following HTTP Headers:

```
HTTP/1.1 200 OK
Server: CouchDB/1.0.1 (Erlang OTP/R13B)
Etag: "7-a19a1a5ecd946dad70e85233ba039ab2"
Date: Fri, 05 Nov 2010 14:54:43 GMT
Content-Type: text/plain;charset=utf-8
Content-Length: 136
Cache-Control: must-revalidate
```

The `Etag` header shows the current revision for the requested document, and the `Content-Length` specifies the length of the data, if the document were requested in full.

Adding any of the query arguments (as supported by `GET``_ method), then the resulting HTTP Headers will correspond to what would be returned. Note that the current revision is not returned when the ``refs_info` argument is used. For example:

```
HTTP/1.1 200 OK
Server: CouchDB/1.0.1 (Erlang OTP/R13B)
Date: Fri, 05 Nov 2010 14:57:16 GMT
Content-Type: text/plain;charset=utf-8
Content-Length: 609
Cache-Control: must-revalidate
```

4.1.4 Creating or updating a document

- **Method**: `PUT /db/doc`
- **Request**: JSON of the new document, or updated version of the existing document
- **Response**: JSON of the document ID and revision
- **Roles permitted**: _writer
- **Query Arguments**:
  - **Argument**: batch
    - **Description**: Allow document store request to be batched with others
    - **Optional**: yes
    - **Type**: string
    - **Supported Values**:
      - **ok**: Enable
- **HTTP Headers**
  - **Header**: `If-Match`

        * **Description**: Current revision of the document for validation
        * **Optional**: yes

- **Return Codes**:
  - **201**: Document has been created successfully
  - **202**: Document accepted for writing (batch mode)

The `PUT` method creates a new named document, or creates a new revision of the existing document. Unlike the `POST` method, you must specify the document ID in the request URL.

For example, to create the document `DocID`, you would send the following request:

```
PUT /db/DocID HTTP/1.1
Accept: application/json

{
  "name": "Hannah",
  "age": 120,
  "gender": "female",
  "_id": "DocID",
  "_attachments": {

  }
}
```

The return type is JSON of the status, document ID,and revision number:

```
{
  "ok": true,
  "id": "DocID",
  "rev": "1-764b9b11845fd0b73cfa0e61acc74ecf"
}
```

Updating an Existing Document

To update an existing document you must specify the current revision number within the `rev` parameter. For example:

```
PUT /db/DocID?rev=1-764b9b11845fd0b73cfa0e61acc74ecf HTTP/1.1
Accept: application/json

{
  "name": "Hannah",
  "age": 40,
  "gender": "female",
  "_id": "DocID",
  "_attachments": {

  },
  "_rev": "1-764b9b11845fd0b73cfa0e61acc74ecf"
}
```

Alternatively, you can supply the current revision number in the `If-Match` HTTP header of the request. For example:

```
PUT /test/DocID
If-Match: 1-61029d20ba39869b1fc879227f5d9f2b
Content-Type: application/json

{
  "name": "Hannah",
  "age": 40,
```

```
  "gender": "female",
  "_id": "DocID",
  "_attachments": {

  },
  "_rev": "1-764b9b11845fd0b73cfa0e61acc74ecf"
}
```

The JSON returned will include the updated revision number:

```
{
  "ok": true,
  "id": "DocID",
  "rev": "2-a537656346d6aa02353e1d31f07b16c4"
}
```

Overriding the default write quorum

The w query-string parameter on updates overrides the default write quorum for the database. When the N copies of each document are written, the client will receive a response after w of them have been committed successfully (the operations to commit the remaining copies will continue in the background). w defaults to the simple majority of N, which is the recommended choice for most applications.

See also

For information on batched writes, which can provide improved performance, see *UUID generation algorithms*.

4.1.5 Deleting a document

- **Method**: `DELETE /db/doc`
- **Request**: None
- **Response**: JSON of the deleted revision
- **Roles permitted**: _writer
- **Query Arguments**:
  - **Argument**: rev
    - **Description**: Current revision of the document for validation
    - **Optional**: yes
    - **Type**: string
- **HTTP Headers**
  - **Header**: `If-Match`
    - **Description**: Current revision of the document for validation
    - **Optional**: yes
- **Return Codes**:
  - **409**: Revision is missing, invalid or not the latest

Deletes the specified document from the database. You must supply the current (latest) revision, either by using the `rev` parameter to specify the revision:

```
DELETE /test/DocID?rev=3-a1a9b39ee3cc39181b796a69cb48521c
Content-Type: application/json
```

Alternatively, you can use ETags with the `If-Match` field:

```
DELETE /test/DocID
If-Match: 3-a1a9b39ee3cc39181b796a69cb48521c
Content-Type: application/json
```

The returned JSON contains the document ID, revision and status:

```
{
   "id" : "DocID",
   "ok" : true,
   "rev" : "4-2719fd41187c60762ff584761b714cfb"
}
```

Note: Note that deletion of a record increments the revision number. The use of a revision for deletion of the record allows replication of the database to correctly track the deletion in synchronized copies.

4.1.6 Copying a document

- **Method**: `COPY /db/doc`
- **Request**: None
- **Response**: JSON of the new document and revision
- **Roles permitted**: _writer
- **Query Arguments**:
  - **Argument**: rev
    - **Description**: Revision to copy from
    - **Optional**: yes
    - **Type**: string
- **HTTP Headers**
  - **Header**: `Destination`
    - **Description**: Destination document (and optional revision)
    - **Optional**: no
- **Return Codes**:
  - **201**: Document has been copied and created successfully
  - **409**: Revision is missing, invalid or not the latest

The `COPY` command (which is non-standard HTTP) copies an existing document to a new or existing document.

The source document is specified on the request line, with the `Destination` HTTP Header of the request specifying the target document.

Copying a Document to a new document

You can copy the latest version of a document to a new document by specifying the current document and target document:

```
COPY /test/DocID
Content-Type: application/json
Destination: NewDocId
```

The above request copies the document `DocID` to the new document `NewDocId`. The response is the ID and revision of the new document.

```
{
   "id" : "NewDocId",
   "rev" : "1-9c65296036141e575d32ba9c034dd3ee"
}
```

Copying from a Specific Revision

To copy *from* a specific version, use the `rev` argument to the query string:

```
COPY /test/DocID?rev=5-acfd32d233f07cea4b4f37daaacc0082
Content-Type: application/json
Destination: NewDocID
```

The new document will be created using the information in the specified revision of the source document.

Copying to an Existing Document

To copy to an existing document, you must specify the current revision string for the target document, using the `rev` parameter to the `Destination` HTTP Header string. For example:

```
COPY /test/DocID
Content-Type: application/json
Destination: ExistingDocID?rev=1-9c65296036141e575d32ba9c034dd3ee
```

The return value will be the new revision of the copied document:

```
{
   "id" : "ExistingDocID",
   "rev" : "2-55b6a1b251902a2c249b667dab1c6692"
}
```

4.2 Attachments

4.2.1 Retrieving an attachment

- **Method**: `GET /db/doc/attachment`
- **Request**: None
- **Response**: Returns the attachment data
- **Roles permitted**: _reader

Returns the file attachment `attachment` associated with the document `doc`. The raw data of the associated attachment is returned (just as if you were accessing a static file. The returned HTTP `Content-type` will be the same as the content type set when the document attachment was submitted into the database.

4.2.2 Creating or updating an attachment

- **Method**: `PUT /db/doc/attachment`
- **Request**: Raw document data
- **Response**: JSON document status
- **Roles permitted**: _writer

- **Query Arguments**:
  - **Argument**: rev
    - **Description**: Current document revision
    - **Optional**: no
    - **Type**: string
- **HTTP Headers**
  - **Header**: `Content-Length`
    - **Description**: Length (bytes) of the attachment being uploaded
    - **Optional**: no
  - **Header**: `Content-Type`
    - **Description**: MIME type for the uploaded attachment
    - **Optional**: no
  - **Header**: `If-Match`
    - **Description**: Current revision of the document for validation
    - **Optional**: yes
- **Return Codes**:
  - **201**: Attachment has been accepted

Upload the supplied content as an attachment to the specified document (`doc`). The `attachment` name provided must be a URL encoded string. You must also supply either the `rev` query argument or the `If-Match` HTTP header for validation, and the HTTP headers (to set the attachment content type). The content type is used when the attachment is requested as the corresponding content-type in the returned document header.

For example, you could upload a simple text document using the following request:

```
PUT /recipes/FishStew/basic?rev=8-a94cb7e50ded1e06f943be5bfbddf8ca
Content-Length: 10
Content-Type: text/plain

Roast it
```

Or by using the `If-Match` HTTP header:

```
PUT /recipes/FishStew/basic
If-Match: 8-a94cb7e50ded1e06f943be5bfbddf8ca
Content-Length: 10
Content-Type: text/plain

Roast it
```

The returned JSON contains the new document information:

```
{
   "id" : "FishStew",
   "ok" : true,
   "rev" : "9-247bb19a41bfd9bfdaf5ee6e2e05be74"
}
```

Note: Uploading an attachment updates the corresponding document revision. Revisions are tracked for the parent document, not individual attachments.

Updating an Existing Attachment

Uploading an attachment using an existing attachment name will update the corresponding stored content of the database. Since you must supply the revision information to add an attachment to a document, this serves as validation to update the existing attachment.

4.2.3 Creating a document with an inline attachment

Inline attachments are just like any other attachment, except that their data is included in the document itself via Base 64 encoding when the document is created or updated.

```
{
  "_id":"attachment_doc",
  "_attachments": {
    "foo.txt": {
      "content_type":"text/plain",
      "data": "VGhpcyBpcyBhIGJhc2U2NCBlbmNvZGVkIHRleHQ="
    }
  }
}
```

4.2.4 Deleting an attachment

- **Method**: `DELETE /db/doc/attachment`
- **Request**: None
- **Response**: JSON status
- **Roles permitted**: _writer
- **Query Arguments**:
  - **Argument**: rev
    - **Description**: Current document revision
    - **Optional**: no
    - **Type**: string
- **HTTP Headers**
  - **Header**: `If-Match`
    - **Description**: Current revision of the document for validation
    - **Optional**: yes
- **Return Codes**:
  - **200**: Attachment deleted successfully
  - **409**: Supplied revision is incorrect or missing

Deletes the attachment `attachment` to the specified `doc`. You must supply the `rev` argument with the current revision to delete the attachment.

For example to delete the attachment `basic` from the recipe `FishStew`:

```
DELETE /db/DocID/my+attachment?rev=2-f29c836d0bedc4b4b95cfaa6d99e95df HTTP/1.1
Accept: application/json
```

The returned JSON contains the updated revision information:

```
{
  "ok": true,
  "id": "DocID",
  "rev": "3-aedfb06537c1d77a087eb295571f7fc9"
}
```

CHAPTER

FIVE

DESIGN DOCUMENT METHODS

Design documents provide the main interface for building an application with Cloudant. The design document defines the views and indexers used to extract information from the database. Design documents are created in the same way as you create other database documents, but the content and definition of the documents is different. Design documents are named using an ID defined with the design document URL path, and this URL can then be used to access the database contents.

Views and lists operate together to provide automated (and formatted) output from your database. Indexers are used with Cloudant's Lucene-based search functions.

5.1 Retrieving a design document

Since design documents are just ordinary documents, there is nothing special about retrieving them. The URL path used for design documents is `/db/_design/design-doc`, where `design-doc` is the name of the design document and `db` is the name of the database.

See *Retrieving a document* for information about retrieving documents.

5.2 Creating or updating a design document

- **Method**: `PUT /db/_design/design-doc`
- **Request**: JSON of the design document
- **Response**: JSON status
- **Roles permitted**: _writer

Upload the specified design document, `design-doc`, to the specified database. Design documents are ordinary documents defining views and indexers in the format summarised in the following table.

- **_id**: Design Document ID
- **_rev**: Design Document Revision
- **views**: View
  - **viewname**: View Definition
    - **map**: Map Function for View
    - **reduce (optional)**: Reduce Function for View
- **indexes**: Indexes
  - **index name**: Index definition
    - **analyzer**: Name of the analyzer to be used or an object with the following fields:
      - **name**: Name of the analyzer

- **stopwords**: An array of stop words. Stop words are words that should not be indexed.

* **index**: Function that handles the indexing

5.2.1 The map function

The function contained in the map field is a Javascript function that is called for each document in the database. The map function takes the document as an argument and optionally calls the `emit` function one or more times to emit pairs of keys and values. The simplest example of a map function is this:

```
function(doc) {
  emit(doc._id, doc);
}
```

The result will be that the view contains every document with the key being the id of the document, effectively creating a copy of the database.

5.2.2 The reduce function

If a view has a reduce function, it is used to produce aggregate results for that view. A reduce function is passed a set of intermediate values and combines them to a single value. Reduce functions must accept, as input, results emitted by its corresponding map function ‘’‘as well as results returned by the reduce function itself’‘’. The latter case is referred to as a ‘’rereduce’‘.

Here is an example of a reduce function:

```
function (key, values, rereduce) {
  return sum(values);
}
```

Reduce functions are passed three arguments in the order ‘’key’‘, ‘’values’‘, and ‘’rereduce’‘.

Reduce functions must handle two cases:

1. When `rereduce` is false:

- `key` will be an array whose elements are arrays of the form `[key,id]`, where `key` is a key emitted by the map function and ‘’id” is that of the document from which the key was generated.
- `values` will be an array of the values emitted for the respective elements in `keys`
- i.e. `reduce([ [key1,id1], [key2,id2], [key3,id3] ], [value1,value2,value3], false)`

2. When `rereduce` is true:

- `key` will be `null`.
- `values` will be an array of values returned by previous calls to the reduce function.
- i.e. `reduce(null, [intermediate1,intermediate2,intermediate3], true)` `

Reduce functions should return a single value, suitable for both the ‘’value” field of the final view and as a member of the ‘’values” array passed to the reduce function.

Often, reduce functions can be written to handle rereduce calls without any extra code, like the summation function above. In that case, the ‘’rereduce” argument can be ignored.

5.2.3 The index function

The function contained in the index field is a Javascript function that is called for each document in the database. It takes the document as a parameter, extracts some data from it and then calls the `index` method to index that data. The `index` method take 3 parameters, where the third parameter is optional. The first parameter is the name of the index. If the special value `"default"` is used, the data is stored in the default index, which is queried if

no index name is specified in the search. The second parameter is the data to be indexed. The third parameter is an object that can contain the fields `store` and `index`. If the `store` field contains the value `yes`, the value will be returned in search results, otherwise, it will only be indexed. The `index` field can have the following values describing whether and how the data is indexed:

- `analyzed`: Index the tokens produced by running the field's value through an analyzer.
- `analyzed_no_norms`: Index the tokens produced by running the field's value through an analyzer, and also separately disable the storing of norms.
- `no`: Do not index the field value.
- `not_analyzed`: Index the field's value without using an analyzer, so it can be searched.
- `not_analyzed_no_norms`: Index the field's value without an analyzer, and also disable the indexing of norms.

Here is an example of a simple index function.

```
function(doc) {
  if (doc.foo) {
    index("default", doc.foo);
  }
}
```

For more information on indexing and searching, see *Searching for documents using Lucene queries*.

For more information on writing views, see *Querying a view*.

5.3 Deleting a design document

- **Method**: `DELETE /db/_design/design-doc`
- **Request**: None
- **Response**: JSON of deleted design document
- **Roles permitted**: _writer
- **Query Arguments**:
  - **Argument**: rev
    - **Description**: Current revision of the document for validation
    - **Optional**: yes
    - **Type**: string
- **HTTP Headers**
  - **Header**: `If-Match`
    - **Description**: Current revision of the document for validation
    - **Optional**: yes
- **Return Codes**:
  - **409**: Supplied revision is incorrect or missing

Delete an existing design document. Deleting a design document also deletes all of the associated view indexes, and recovers the corresponding space on disk for the indexes in question.

To delete, you must specify the current revision of the design document using the `rev` query argument.

For example:

```
DELETE /recipes/_design/recipes?rev=2-ac58d589b37d01c00f45a4418c5a15a8
Content-Type: application/json
```

The response contains the delete document ID and revision:

```
{
   "id" : "recipe/_design/recipes"
   "ok" : true,
   "rev" : "3-7a05370bff53186cb5d403f861aca154",
}
```

5.4 Copying a design document

- **Method**: `COPY /db/_design/design-doc`
- **Request**: None
- **Response**: JSON of the new document and revision
- **Roles permitted**: _writer
- **Query Arguments**:
  - **Argument**: rev
    - **Description**: Revision to copy from
    - **Optional**: yes
    - **Type**: string
- **HTTP Headers**
  - **Header**: `Destination`
    - **Description**: Destination document (and optional revision)
    - **Optional**: no

The `COPY` command (non-standard HTTP) copies an existing design document to a new or existing document.

The source design document is specified on the request line, with the `Destination` HTTP Header of the request specifying the target document.

5.4.1 Copying a Design Document

To copy the latest version of a design document to a new document you specify the base document and target document:

```
COPY /recipes/_design/recipes
Content-Type: application/json
Destination: /recipes/_design/recipelist
```

The above request copies the design document `recipes` to the new design document `recipelist`. The response is the ID and revision of the new document.

```
{
   "id" : "recipes/_design/recipelist"
   "rev" : "1-9c65296036141e575d32ba9c034dd3ee",
}
```

Note: Copying a design document does not automatically reconstruct the view indexes. These will be recreated, as with other views, the first time the new view is accessed.

5.4.2 Copying from a Specific Revision

To copy *from* a specific version, use the `rev` argument to the query string:

```
COPY /recipes/_design/recipes?rev=1-e23b9e942c19e9fb10ff1fde2e50e0f5
Content-Type: application/json
Destination: recipes/_design/recipelist
```

The new design document will be created using the specified revision of the source document.

5.4.3 Copying to an Existing Design Document

To copy to an existing document, you must specify the current revision string for the target document, using the `rev` parameter to the `Destination` HTTP Header string. For example:

```
COPY /recipes/_design/recipes
Content-Type: application/json
Destination: recipes/_design/recipelist?rev=1-9c65296036141e575d32ba9c034dd3ee
```

The return value will be the new revision of the copied document:

```
{
   "id" : "recipes/_design/recipes"
   "rev" : "2-55b6a1b251902a2c249b667dab1c6692",
}
```

5.5 Retrieving information about a design document

- **Method**: `GET /db/_design/design-doc/_info`
- **Request**: None
- **Response**: JSON of the design document information
- **Roles permitted**: _reader

Obtains information about a given design document, including the index, index size and current status of the design document and associated index information.

For example, to get the information for the `recipes` design document:

```
GET /recipes/_design/recipes/_info
Content-Type: application/json
```

This returns the following JSON structure:

```
{
   "name" : "recipes"
   "view_index" : {
      "compact_running" : false,
      "updater_running" : false,
      "language" : "javascript",
      "purge_seq" : 10,
      "waiting_commit" : false,
      "waiting_clients" : 0,
      "signature" : "fc65594ee76087a3b8c726caf5b40687",
      "update_seq" : 375031,
      "disk_size" : 16491
   },
}
```

The individual fields in the returned JSON structure are detailed below:

- **name**: Name/ID of Design Document
- **view_index**: View Index
  - **compact_running**: Indicates whether a compaction routine is currently running on the view
  - **disk_size**: Size in bytes of the view as stored on disk
  - **language**: Language for the defined views
  - **purge_seq**: The purge sequence that has been processed
  - **signature**: MD5 signature of the views for the design document
  - **update_seq**: The update sequence of the corresponding database that has been indexed
  - **updater_running**: Indicates if the view is currently being updated
  - **waiting_clients**: Number of clients waiting on views from this design document
  - **waiting_commit**: Indicates if there are outstanding commits to the underlying database that need to processed

5.6 Querying a view

- **Method**: `GET /db/_design/design-doc/_view/view-name`
- **Request**: None
- **Response**: JSON of the documents returned by the view
- **Roles permitted**: _reader
- **Query Arguments**:
  - **Argument**: descending
    - **Description**: Return the documents in descending by key order
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: false
  - **Argument**: endkey
    - **Description**: Stop returning records when the specified key is reached
    - **Optional**: yes
    - **Type**: string
  - **Argument**: endkey_docid
    - **Description**: Stop returning records when the specified document ID is reached
    - **Optional**: yes
    - **Type**: string
  - **Argument**: group
    - **Description**: Group the results using the reduce function to a group or single row
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: false
  - **Argument**: group_level

    - **Description**: Specify the group level to be used
    - **Optional**: yes
    - **Type**: numeric
- **Argument**: include_docs
    - **Description**: Include the full content of the documents in the return
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: false
- **Argument**: inclusive_end
    - **Description**: Specifies whether the specified end key should be included in the result
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: true
- **Argument**: key
    - **Description**: Return only documents that match the specified key
    - **Optional**: yes
    - **Type**: string
- **Argument**: limit
    - **Description**: Limit the number of the returned documents to the specified number
    - **Optional**: yes
    - **Type**: numeric
- **Argument**: reduce
    - **Description**: Use the reduction function
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: true
- **Argument**: skip
    - **Description**: Skip this number of records before starting to return the results
    - **Optional**: yes
    - **Type**: numeric
    - **Default**: 0
- **Argument**: stale
    - **Description**: Allow the results from a stale view to be used
    - **Optional**: yes
    - **Type**: string
    - **Default**:
    - **Supported Values**
        - **ok**: Allow stale views
- **Argument**: startkey

        * **Description**: Return records starting with the specified key
        * **Optional**: yes
        * **Type**: string
    - **Argument**: startkey_docid
        * **Description**: Return records starting with the specified document ID
        * **Optional**: yes
        * **Type**: string
    - **Argument**: update_seq
        * **Description**: Include the update sequence in the generated results
        * **Optional**: yes
        * **Type**: boolean
        * **Default**: false

Executes the specified `view-name` from the specified `design-doc` design document.

5.6.1 Querying Views and Indexes

The definition of a view within a design document also creates an index based on the key information defined within each view. The production and use of the index significantly increases the speed of access and searching or selecting documents from the view.

However, the index is not updated when new documents are added or modified in the database. Instead, the index is generated or updated, either when the view is first accessed, or when the view is accessed after a document has been updated. In each case, the index is updated before the view query is executed against the database.

View indexes are updated incrementally in the following situations:

- A new document has been added to the database.
- A document has been deleted from the database.
- A document in the database has been updated.

View indexes are rebuilt entirely when the view definition changes. To achieve this, a 'fingerprint' of the view definition is created when the design document is updated. If the fingerprint changes, then the view indexes are entirely rebuilt. This ensures that changes to the view definitions are reflected in the view indexes.

Note: View index rebuilds occur when one view from the same the view group (i.e. all the views defined within a single a design document) has been determined as needing a rebuild. For example, if if you have a design document with different views, and you update the database, all three view indexes within the design document will be updated.

Because the view is updated when it has been queried, it can result in a delay in returned information when the view is accessed, especially if there are a large number of documents in the database and the view index does not exist. There are a number of ways to mitigate, but not completely eliminate, these issues. These include:

- Create the view definition (and associated design documents) on your database before allowing insertion or updates to the documents. If this is allowed while the view is being accessed, the index can be updated incrementally.
- Manually force a view request from the database. You can do this either before users are allowed to use the view, or you can access the view manually after documents are added or updated.
- Use the `/db/_changes` method to monitor for changes to the database and then access the view to force the corresponding view index to be updated. See *Obtaining a list of changes* for more information.

None of these can completely eliminate the need for the indexes to be rebuilt or updated when the view is accessed, but they may lessen the effects on end-users of the index update affecting the user experience.

Another alternative is to allow users to access a 'stale' version of the view index, rather than forcing the index to be updated and displaying the updated results. Using a stale view may not return the latest information, but will return the results of the view query using an existing version of the index.

For example, to access the existing stale view `by_recipe` in the `recipes` design document:

```
/recipes/_design/recipes/_view/by_recipe?stale=ok
```

Accessing a stale view:

- Does not trigger a rebuild of the view indexes, even if there have been changes since the last access.
- Returns the current version of the view index, if a current version exists.
- Returns an empty result set if the given view index does exist.

As an alternative, you use the `update_after` value to the `stale` parameter. This causes the view to be returned as a stale view, but for the update process to be triggered after the view information has been returned to the client.

In addition to using stale views, you can also make use of the `update_seq` query argument. Using this query argument generates the view information including the update sequence of the database from which the view was generated. The returned value can be compared this to the current update sequence exposed in the database information (returned by *Retrieving information about a database*).

5.6.2 Sorting Returned Rows

Each element within the returned array is sorted using native UTF-8 sorting according to the contents of the key portion of the emitted content. The basic order of output is as follows:

- `null`
- `false`
- `true`
- Numbers
- Text (case sensitive, lowercase first)
- Arrays (according to the values of each element, in order)
- Objects (according to the values of keys, in key order)

You can reverse the order of the returned view information by using the `descending` query value set to true. For example, Retrieving the list of recipes using the `by_title` (limited to 5 records) view:


```
{
   "offset" : 0,
   "rows" : [
      {
         "id" : "3-tiersalmonspinachandavocadoterrine",
         "key" : "3-tier salmon, spinach and avocado terrine",
         "value" : [
            null,
            "3-tier salmon, spinach and avocado terrine"
         ]
      },
      {
         "id" : "Aberffrawcake",
         "key" : "Aberffraw cake",
         "value" : [
            null,
            "Aberffraw cake"
         ]
```

```
      },
      {
         "id" : "Adukiandorangecasserole-microwave",
         "key" : "Aduki and orange casserole - microwave",
         "value" : [
            null,
            "Aduki and orange casserole - microwave"
         ]
      },
      {
         "id" : "Aioli-garlicmayonnaise",
         "key" : "Aioli - garlic mayonnaise",
         "value" : [
            null,
            "Aioli - garlic mayonnaise"
         ]
      },
      {
         "id" : "Alabamapeanutchicken",
         "key" : "Alabama peanut chicken",
         "value" : [
            null,
            "Alabama peanut chicken"
         ]
      }
   ],
   "total_rows" : 2667
}
```


Requesting the same in descending order will reverse the entire view content. For example the request

```
GET /recipes/_design/recipes/_view/by_title?limit=5&descending=true
Accept: application/json
Content-Type: application/json
```

Returns the last 5 records from the view:


```
{
   "offset" : 0,
   "rows" : [
      {
         "id" : "Zucchiniinagrodolcesweet-sourcourgettes",
         "key" : "Zucchini in agrodolce (sweet-sour courgettes)",
         "value" : [
            null,
            "Zucchini in agrodolce (sweet-sour courgettes)"
         ]
      },
      {
         "id" : "Zingylemontart",
         "key" : "Zingy lemon tart",
         "value" : [
            null,
            "Zingy lemon tart"
         ]
      },
      {
         "id" : "Zestyseafoodavocado",
         "key" : "Zesty seafood avocado",
         "value" : [
            null,
            "Zesty seafood avocado"
         ]
```

```
        },
        {
            "id" : "Zabaglione",
            "key" : "Zabaglione",
            "value" : [
                null,
                "Zabaglione"
            ]
        },
        {
            "id" : "Yogurtraita",
            "key" : "Yogurt raita",
            "value" : [
                null,
                "Yogurt raita"
            ]
        }
    ],
    "total_rows" : 2667
}
```

The sorting direction is applied before the filtering applied using the `startkey` and `endkey` query arguments. For example the following query:

```
GET /recipes/_design/recipes/_view/by_ingredient?startkey=%22carrots%22&endkey=%22egg%22
Accept: application/json
Content-Type: application/json
```

Will operate correctly when listing all the matching entries between "carrots" and `egg`. If the order of output is reversed with the `descending` query argument, the view request will return no entries:

```
GET /recipes/_design/recipes/_view/by_ingredient?descending=true&startkey=%22carrots%22&endkey=%2
Accept: application/json
Content-Type: application/json
```

The returned result is empty:

```
{
    "total_rows" : 26453,
    "rows" : [],
    "offset" : 21882
}
```

The results will be empty because the entries in the view are reversed before the key filter is applied, and therefore the `endkey` of "egg" will be seen before the `startkey` of "carrots", resulting in an empty list.

Instead, you should reverse the values supplied to the `startkey` and `endkey` parameters to match the descending sorting applied to the keys. Changing the previous example to:

```
GET /recipes/_design/recipes/_view/by_ingredient?descending=true&startkey=%22egg%22&endkey=%22car
Accept: application/json
Content-Type: application/json
```

5.6.3 Specifying Start and End Values

The `startkey` and `endkey` query arguments can be used to specify the range of values to be displayed when querying the view.

5.7 Querying a view using a list of keys

- **Method**: `POST /db/_design/design-doc/_view/view-name`

- **Request**: List of keys to be returned from specified view
- **Response**: JSON of the documents returned by the view
- **Roles permitted**: _reader
- **Query Arguments**:
  - **Argument**: descending
    - **Description**: Return the documents in descending by key order
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: false
  - **Argument**: endkey
    - **Description**: Stop returning records when the specified key is reached
    - **Optional**: yes
    - **Type**: string
  - **Argument**: endkey_docid
    - **Description**: Stop returning records when the specified document ID is reached
    - **Optional**: yes
    - **Type**: string
  - **Argument**: group
    - **Description**: Group the results using the reduce function to a group or single row
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: false
  - **Argument**: group_level
    - **Description**: Specify the group level to be used
    - **Optional**: yes
    - **Type**: numeric
  - **Argument**: include_docs
    - **Description**: Include the full content of the documents in the return
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: false
  - **Argument**: inclusive_end
    - **Description**: Specifies whether the specified end key should be included in the result
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: true
  - **Argument**: key
    - **Description**: Return only documents that match the specified key
    - **Optional**: yes

    - **Type**: string
- **Argument**: limit
    - **Description**: Limit the number of the returned documents to the specified number
    - **Optional**: yes
    - **Type**: numeric
- **Argument**: reduce
    - **Description**: Use the reduction function
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: true
- **Argument**: skip
    - **Description**: Skip this number of records before starting to return the results
    - **Optional**: yes
    - **Type**: numeric
    - **Default**: 0
- **Argument**: stale
    - **Description**: Allow the results from a stale view to be used
    - **Optional**: yes
    - **Type**: string
    - **Default**:
    - **Supported Values**:
        - **ok**: Allow stale views
- **Argument**: startkey
    - **Description**: Return records starting with the specified key
    - **Optional**: yes
    - **Type**: string
- **Argument**: startkey_docid
    - **Description**: Return records starting with the specified document ID
    - **Optional**: yes
    - **Type**: string
- **Argument**: update_seq
    - **Description**: Include the update sequence in the generated results
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: false

Executes the specified `view-name` from the specified `design-doc` design document. Unlike the `GET` method for accessing views, the `POST` method supports the specification of explicit keys to be retrieved from the view results. The remainder of the `POST` view functionality is identical to the *Querying a view* API.

For example, the request below will return all the recipes where the key for the view matches either "claret" or "clear apple cider" :

```
POST /recipes/_design/recipes/_view/by_ingredient
Content-Type: application/json

{
   "keys" : [
      "claret",
      "clear apple juice"
   ]
}
```

The returned view data contains the standard view information, but only where the keys match.

```
{
   "total_rows" : 26484,
   "rows" : [
      {
         "value" : [
            "Scotch collops"
         ],
         "id" : "Scotchcollops",
         "key" : "claret"
      },
      {
         "value" : [
            "Stand pie"
         ],
         "id" : "Standpie",
         "key" : "clear apple juice"
      }
   ],
   "offset" : 6324
}
```

5.7.1 Multi-document Fetching

By combining the `POST` method to a given view with the `include_docs=true` query argument you can obtain multiple documents from a database. The result is more efficient than using multiple *Retrieving a document* requests.

For example, sending the following request for ingredients matching "claret" and "clear apple juice":

```
POST /recipes/_design/recipes/_view/by_ingredient?include_docs=true
Content-Type: application/json

{
   "keys" : [
      "claret",
      "clear apple juice"
   ]
}
```

Returns the full document for each recipe:

```
{
   "offset" : 6324,
   "rows" : [
      {
         "doc" : {
            "_id" : "Scotchcollops",
            "_rev" : "1-bcbdf724f8544c89697a1cbc4b9f0178",
            "cooktime" : "8",
            "ingredients" : [
```

```
                {
                   "ingredient" : "onion",
                   "ingredtext" : "onion, peeled and chopped",
                   "meastext" : "1"
                },
             ...
             ],
             "keywords" : [
                "cook method.hob, oven, grill@hob",
                "diet@wheat-free",
                "diet@peanut-free",
                "special collections@classic recipe",
                "cuisine@british traditional",
                "diet@corn-free",
                "diet@citrus-free",
                "special collections@very easy",
                "diet@shellfish-free",
                "main ingredient@meat",
                "occasion@christmas",
                "meal type@main",
                "diet@egg-free",
                "diet@gluten-free"
             ],
             "preptime" : "10",
             "servings" : "4",
             "subtitle" : "This recipe comes from an old recipe book of 1683 called 'The Gentlewoma
             "title" : "Scotch collops",
             "totaltime" : "18"
          },
          "id" : "Scotchcollops",
          "key" : "claret",
          "value" : [
             "Scotch collops"
          ]
       },
       {
          "doc" : {
             "_id" : "Standpie",
             "_rev" : "1-bff6edf3ca2474a243023f2dad432a5a",
             "cooktime" : "92",
             "ingredients" : [
...             ],
             "keywords" : [
                "diet@dairy-free",
                "diet@peanut-free",
                "special collections@classic recipe",
                "cuisine@british traditional",
                "diet@corn-free",
                "diet@citrus-free",
                "occasion@buffet party",
                "diet@shellfish-free",
                "occasion@picnic",
                "special collections@lunchbox",
                "main ingredient@meat",
                "convenience@serve with salad for complete meal",
                "meal type@main",
                "cook method.hob, oven, grill@hob / oven",
                "diet@cow dairy-free"
             ],
             "preptime" : "30",
             "servings" : "6",
             "subtitle" : "Serve this pie with pickled vegetables and potato salad.",
             "title" : "Stand pie",
```

```
            "totaltime" : "437"
         },
         "id" : "Standpie",
         "key" : "clear apple juice",
         "value" : [
            "Stand pie"
         ]
      }
   ],
   "total_rows" : 26484
}
```

5.8 Searching for documents using Lucene queries

- **Method**: `GET /db/_design/design-doc/_search/search-name`
- **Request Body**: None
- **Response Body**: Returns the result of the search
- **Roles permitted**: _reader
- **Query Arguments**:
  - **Argument**: query
    - **Description**: A Lucene query.
    - **Optional**: no
    - **Type**: string or number
  - **Argument**: bookmark
    - **Description**: A bookmark that was received from a previous search. This allows you to page through the results. If there are no more results after the bookmark, you will get a response with an empty rows array and the same bookmark. That way you can determine that you have reached the end of the result list.
    - **Optional**: yes
    - **Type**: string
  - **Argument**: stale
    - **Description**: Allow the results from a stale view to be used
    - **Optional**: yes
    - **Type**: string
    - **Default**:
    - **Supported Values**:
      - **ok**: Allow stale views
  - **Argument**: limit
    - **Description**: Limit the number of the returned documents to the specified number
    - **Optional**: yes
    - **Type**: numeric
  - **Argument**: include_docs
    - **Description**: Include the full content of the documents in the return
    - **Optional**: yes

        * **Type**: boolean
        * **Default**: false
    - **Argument**: sort
        * **Description**: Specifies the sort order of the results.
        * **Optional**: yes
        * **Type**: JSON
        * **Supported Values**:
            - A JSON string of the form `"fieldname<type>"` or `-fieldname<type>` for descending order, where fieldname is the name of a string or number field and type is either `number` or `string`. The type part is optional and defaults to number. Some examples are `"foo"`, `"-foo"`, `"bar<string>"`, `"-foo<number>"`. String fields used for sorting must not be analyzed fields. The field(s) used for sorting must be indexed by the same indexer used for the search query.
            - A JSON array of such strings.

This request searches for documents whose index fields match the Lucene query. Which fields of a document are indexed and how is determined by the index functions in the design document. For more information, see *Creating or updating a design document*

Here is an example of an HTTP request:

```
GET /db/_design/my+searches/_search/bar?q=a*&sort=["foo<number>"] HTTP/1.1
Accept: application/json
```

5.8.1 Search Response

The response is a JSON document that has the following structure.

- `total_rows`: Number of results returned
- `bookmark`: String to be submitted in the next query to page through results. If this response contained no results, the bookmark will be the same as the one used to obtain this response.
- `rows`: Array of objects describing a search result
    - `id`: Document ID
    - `order`: Specifies the order with regard to the indexed fields
    - `fields`: Object containing other search indexes

Here is the response corresponding to the request above:

```
{
  "total_rows": 3,
  "bookmark": "g1AAAACWeJzLYWBgYMpgTmFQSElKzi9KdUhJMtbLTS3KLElMT9VLzskvTUnMK9HLSy3JAalMcgCSSfX____
  "rows": [{
    "id": "dd828eb4-c3f1-470f-aeff-c375ef70e4ad",
    "order": [0.0, 1],
    "fields": {
      "default": "aa",
      "foo": 0.0
    }
  }, {
    "id": "ea522cf1-eb8e-4477-aa92-d1fa459bb216",
    "order": [1.0, 0],
    "fields": {
      "default": "ab",
      "foo": 1.0
```

```
    }
  }, {
    "id": "c838baed-d573-43ea-9c34-621cf0f13301",
    "order": [2.0, 0],
    "fields": {
      "default": "ac",
      "foo": 2.0
    }
  }]
}
```

CHAPTER

SIX

MISCELLANEOUS METHODS

The miscellaneous interface methods provide the basic interface for obtaining server information and getting and setting configuration information.

A list of the available methods and URL paths is provided below:

Method	Path	Description
GET	/	Get the welcome message and version information
GET	/_active_tasks	Obtain a list of the tasks running in the server
GET	/_all_dbs	Get a list of all the DBs
POST	/_replicate	Set or cancel replication
GET	/_uuids	Get generated UUIDs from the server

6.1 Retrieving information about the server

- **Method**: `GET /`
- **Request**: None
- **Response**: Welcome message and version
- **Return Codes**:
  - **200**: Request completed successfully.

Accessing the root returns meta information about the server. The response is a JSON structure containing information about the server, including a welcome message and the version of the server. The server version describes the CouchDB version the server is compatible with, whereas the cloudant_build is the build number of Cloudant's CouchDb implementation.

```
{
  "couchdb": "Welcome",
  "version": "1.0.2",
  "cloudant_build":"1138"
}
```

6.2 Retrieving a list of active tasks

- **Method**: `GET /_active_tasks`
- **Request**: None
- **Response**: List of running tasks, including the task type, name, status and process ID
- **Roles permitted**: _admin
- **Return Codes**:

- **200**: Request completed successfully.

You can obtain a list of active tasks by using the `/_active_tasks` URL. The result is a JSON array of the currently running tasks, with each task being described with a single object. For example:


```
[
  {
    "user": null,
    "updated_on": 1363274088,
    "type": "replication",
    "target": "https://repl:*****@tsm.cloudant.com/user-3dglstqg8aq0uunzimv4uiimy/",
    "docs_read": 0,
    "doc_write_failures": 0,
    "doc_id": "tsm-admin__to__user-3dglstqg8aq0uunzimv4uiimy",
    "continuous": true,
    "checkpointed_source_seq": "403-g1AAAADfeJzLYWBgYMlgTmGQS0lKzi9KdUhJMjTRyyrNSS3QS87JL01JzCvRy
    "changes_pending": 134,
    "pid": "<0.1781.4101>",
    "node": "dbcore@db11.julep.cloudant.net",
    "docs_written": 0,
    "missing_revisions_found": 0,
    "progress": 75,
    "replication_id": "d0cdbfee50a80fd43e83a9f62ea650ad+continuous",
    "revisions_checked": 0,
    "source": "https://repl:*****@tsm.cloudant.com/tsm-admin/",
    "source_seq": "537-g1AAAADfeJzLYWBgYMlgTmGQS0lKzi9KdUhJMjTUyyrNSS3QS87JL01JzCvRy0styQGqY0pkSL
    "started_on": 1363274083
  },
  {
    "user": "acceptly",
    "updated_on": 1363273779,
    "type": "indexer",
    "node": "dbcore@db11.julep.cloudant.net",
    "pid": "<0.20723.4070>",
    "changes_done": 189,
    "database": "shards/00000000-3fffffff/acceptly/acceptly_my_chances_logs_live.1321035717",
    "design_document": "_design/MyChancesLogCohortReport",
    "progress": 0,
    "started_on": 1363273094,
    "total_changes": 26389
  },
  {
    "view": 1,
    "user": "acceptly",
    "updated_on": 1363273504,
    "type": "view_compaction",
    "total_changes": 26095,
    "node": "dbcore@db11.julep.cloudant.net",
    "pid": "<0.21218.4070>",
    "changes_done": 20000,
    "database": "shards/80000000-bfffffff/acceptly/acceptly_my_chances_logs_live.1321035717",
    "design_document": "_design/MyChancesLogCohortReport",
    "phase": "view",
    "progress": 76,
    "started_on": 1363273094
  },
  {
    "updated_on": 1363274040,
    "node": "dbcore@db11.julep.cloudant.net",
    "pid": "<0.29256.4053>",
    "changes_done": 272195,
    "database": "shards/00000000-3fffffff/heroku/app3245179/id_f21a08b7005e_logs.1346083461",
    "progress": 100,
    "started_on": 1363272496,
```

```
    "total_changes": 272195,
    "type": "database_compaction"
  }
]
```

The returned structure includes the following fields for each task:

- **tasks** [array]: Active Task
  - **pid**:Process ID
  - **status**: Task status message
  - **task**: Task name
  - **type**: Operation Type

For the task's type, valid values include:

- `database_compaction`
- `replication`
- `view_compaction`
- `indexer`

You can find an example for each one above.

6.3 Replicating a database

- **Method**: `POST /_replicate`
- **Request**: Replication specification
- **Response**: TBD
- **Roles permitted**: _admin
- **Return Codes**:
  - **200**: Replication request successfully completed
  - **202**: Continuous replication request has been accepted
  - **404**: Either the source or target DB is not found
  - **500**: JSON specification was invalid

Request, configure, or stop, a replication operation.

The specification of the replication request is controlled through the JSON content of the request. The JSON should be an object with the fields defining the source, target and other options. The fields of the JSON request are shown in the table below:

- **cancel (optional)**: Cancels the replication
- **continuous (optional)**: Configure the replication to be continuous
- **create_target (optional)**: Creates the target database
- **doc_ids (optional)**: Array of document IDs to be synchronized
- **proxy (optional)**: Address of a proxy server through which replication should occur
- **source**: Source database URL
- **target**: Target database URL

6.3.1 Replication Operation

The aim of the replication is that at the end of the process, all active documents on the source database are also in the destination database and all documents that were deleted in the source databases are also deleted (if they exist) on the destination database.

Replication can be described as either push or pull replication:

- *Pull replication* is where the `source` is the remote database instance, and the `destination` is the local database.

  Pull replication is the most useful solution to use if your source database has a permanent IP address, and your destination (local) database may have a dynamically assigned IP address (for example, through DHCP). This is particularly important if you are replicating to a mobile or other device from a central server.

- *Push replication* is where the `source` is a local database, and `destination` is a remote database.

For example, to request replication between a database on the server example.com, and a database on Cloudant you might use the following request:

```
POST /_replicate
Content-Type: application/json
Accept: application/json

{
   "source" : "http://example.com/recipes",
   "target" : "http://username.cloudant.com/recipes",
}
```

In all cases, the requested databases in the `source` and `target` specification must exist. If they do not, an error will be returned within the JSON object:

```
{
   "error" : "db_not_found"
   "reason" : "could not open http://username.cloudant.com/ol1ka/",
}
```

You can create the target database (providing your user credentials allow it) by adding the `create_target` field to the request object:

```
POST http://username.cloudant.com/_replicate
Content-Type: application/json
Accept: application/json

{
   "create_target" : true
   "source" : "http://example.com/recipes",
   "target" : "http://username.cloudant.com/recipes",
}
```

The `create_target` field is not destructive. If the database already exists, the replication proceeds as normal.

6.3.2 Single Replication

You can request replication of a database so that the two databases can be synchronized. By default, the replication process occurs one time and synchronizes the two databases together. For example, you can request a single synchronization between two databases by supplying the `source` and `target` fields within the request JSON content.

```
POST /_replicate
Content-Type: application/json
Accept: application/json

{
```

```
   "source" : "http://username.cloudant.com/recipes",
   "target" : "http://username.cloudant.com/recipes-snapshot",
}
```

In the above example, the databases `recipes` and `recipes-snapshot` will be synchronized. The response will be a JSON structure containing the success (or failure) of the synchronization process, and statistics about the process:

```
{
   "ok" : true,
   "history" : [
      {
         "docs_read" : 1000,
         "session_id" : "52c2370f5027043d286daca4de247db0",
         "recorded_seq" : 1000,
         "end_last_seq" : 1000,
         "doc_write_failures" : 0,
         "start_time" : "Thu, 28 Oct 2010 10:24:13 GMT",
         "start_last_seq" : 0,
         "end_time" : "Thu, 28 Oct 2010 10:24:14 GMT",
         "missing_checked" : 0,
         "docs_written" : 1000,
         "missing_found" : 1000
      }
   ],
   "session_id" : "52c2370f5027043d286daca4de247db0",
   "source_last_seq" : 1000
}
```

The structure defines the replication status, as described in the table below:

- **history [array]**: Replication History
  - **doc_write_failures**: Number of document write failures
  - **docs_read**: Number of documents read
  - **docs_written**: Number of documents written to target
  - **end_last_seq**: Last sequence number in changes stream
  - **end_time**: Date/Time replication operation completed
  - **missing_checked**: Number of missing documents checked
  - **missing_found**: Number of missing documents found
  - **recorded_seq**: Last recorded sequence number
  - **session_id**: Session ID for this replication operation
  - **start_last_seq**: First sequence number in changes stream
  - **start_time**: Date/Time replication operation started
- **ok**: Replication status
- **session_id**: Unique session ID
- **source_last_seq**: Last sequence number read from source database

6.3.3 Continuous Replication

Synchronization of a database with the previously noted methods happens only once, at the time the replicate request is made. To have the target database permanently replicated from the source, you must set the `continuous` field of the JSON object within the request to true.

With continuous replication changes in the source database are replicated to the target database in perpetuity until you specifically request that replication ceases.

```
POST /_replicate
Content-Type: application/json
Accept: application/json

{
   "continuous" : true
   "source" : "http://example.com/recipes",
   "target" : "http://username.cloudant.com/recipes",
}
```

Changes will be replicated between the two databases as long as a network connection is available between the two instances.

Note: Two keep two databases synchronized with each other, you need to set replication in both directions; that is, you must replicate from `databasea` to `databaseb`, and separately from `databaseb` to `databasea`.

6.3.4 Canceling Continuous Replication

You can cancel continuous replication by adding the `cancel` field to the JSON request object and setting the value to true. Note that the structure of the request must be identical to the original for the cancellation request to be honoured. For example, if you requested continuous replication, the cancellation request must also contain the `continuous` field.

For example, the replication request:

```
POST /_replicate
Content-Type: application/json
Accept: application/json

{
   "source" : "http://example.com/recipes",
   "target" : "http://username.cloudant.com/recipes",
   "create_target" : true,
   "continuous" : true
}
```

Must be canceled using the request:

```
POST /_replicate
Content-Type: application/json
Accept: application/json

{
    "cancel" : true,
    "continuous" : true
    "create_target" : true,
    "source" : "http://example.com/recipes",
    "target" : "http://username.cloudant.com/recipes",
}
```

Requesting cancellation of a replication that does not exist results in a 404 error.

6.4 Retrieving UUIDs

- **Method**: `GET /_uuids`
- **Request**: None

- **Response**: List of UUIDs
- **Query Arguments**:
  - **Argument**: count
    - **Description**: Number of UUIDs to return
    - **Optional**: yes
    - **Type**: numeric
- **Return Codes**:
  - **200**: Request completed successfully.

Requests one or more Universally Unique Identifiers (UUIDs). The response is a JSON object providing a list of UUIDs. For example:

```
{
   "uuids" : [
      "7e4b5a14b22ec1cf8e58b9cdd0000da3"
   ]
}
```

You can use the `count` argument to specify the number of UUIDs to be returned. For example:

```
GET /_uuids?count=5
```

Returns:

```
{
   "uuids" : [
      "c9df0cdf4442f993fc5570225b405a80",
      "c9df0cdf4442f993fc5570225b405bd2",
      "c9df0cdf4442f993fc5570225b405e42",
      "c9df0cdf4442f993fc5570225b4061a0",
      "c9df0cdf4442f993fc5570225b406a20"
   ]
}
```

For example, changing the UUID type to `random`:

```
PUT /_config/uuids/algorithm
Content-Type: application/json
Accept: */*

"random"
```

When obtaining a list of UUIDs:

```
{
   "uuids" : [
      "031aad7b469956cf2826fcb2a9260492",
      "6ec875e15e6b385120938df18ee8e496",
      "cff9e881516483911aa2f0e98949092d",
      "b89d37509d39dd712546f9510d4a9271",
      "2e0dbf7f6c4ad716f21938a016e4e59f"
   ]
}
```

CHAPTER

SEVEN

LOCAL (NON-REPLICATING) DOCUMENT METHODS

The Local (non-replicating) document interface allows you to create local documents that are not replicated to other databases. These documents can be used to hold configuration or other information that is required specifically on the local server instance.

Local documents have the following limitations:

- Local documents are not replicated to other databases.
- The ID of the local document must be known for the document to accessed. You cannot obtain a list of local documents from the database.
- Local documents are not output by views, or the `_all_docs` view.

Local documents can be used when you want to store configuration or other information for the current (local) instance of a given database.

A list of the available methods and URL paths are provided below:

Method	Path	Description
GET	/db/_local/local-doc	Returns the latest revision of the non-replicated document
PUT	/db/_local/local-doc	Inserts a new version of the non-replicated document
DELETE	/db/_local/local-doc	Deletes the non-replicated document
COPY	/db/_local/local-doc	Copies the non-replicated document

7.1 Retrieving a local document

- **Method**: `GET /db/_local/local-doc`
- **Request**: None
- **Response**: JSON of the returned document
- **Roles permitted**: _reader
- **Query Arguments**:
  - **Argument**: rev
    - **Description**: Specify the revision to return
    - **Optional**: yes
    - **Type**: string
    - **Supported Values**:
      - **true**: Includes the revisions
  - **Argument**: revs

        - **Description**: Return a list of the revisions for the document
        - **Optional**: yes
        - **Type**: boolean
    - **Argument**: revs_info
        - **Description**: Return a list of detailed revision information for the document
        - **Optional**: yes
        - **Type**: boolean
        - **Supported Values**
            - **true**: Includes the revisions
- **Return Codes**:
    - **400**: The format of the request or revision was invalid
    - **404**: The specified document or revision cannot be found, or has been deleted

Gets the specified local document. The semantics are identical to accessing a standard document in the specified database, except that the document is not replicated. See *Retrieving a document*.

7.2 Creating or updating a local document

- **Method**: `PUT /db/_local/local-doc`
- **Request**: JSON of the document
- **Response**: JSON with the committed document information
- **Roles permitted**: _writer
- **Return Codes**:
    - **201**: Document has been created successfully

Stores the specified local document. The semantics are identical to storing a standard document in the specified database, except that the document is not replicated. See *Creating or updating a document*.

7.3 Deleting a local document

- **Method**: `DELETE /db/_local/local-doc`
- **Request**: None
- **Response**: JSON with the deleted document information
- **Roles permitted**: _writer
- **Query Arguments**:
    - **Argument**: rev
        - **Description**: Current revision of the document for validation
        - **Optional**: yes
        - **Type**: string
- **HTTP Headers**
    - **Header**: `If-Match`
        - **Description**: Current revision of the document for validation

        * **Optional**: yes

- **Return Codes**:
  - **409**: Supplied revision is incorrect or missing

Deletes the specified local document. The semantics are identical to deleting a standard document in the specified database, except that the document is not replicated. See *Deleting a document*.

7.4 Copying a local document

- **Method**: `COPY /db/_local/local-doc`
- **Request**: None
- **Response**: JSON of the copied document
- **Roles permitted**: _writer
- **Query Arguments**:
  - **Argument**: rev
    * **Description**: Revision to copy from
    * **Optional**: yes
    * **Type**: string
- **HTTP Headers**
  - **Header**: `Destination`
    * **Description**: Destination document (and optional revision)
    * **Optional**: no

Copies the specified local document. The semantics are identical to copying a standard document in the specified database, except that the document is not replicated. See *Copying a document*.