Apache CouchDB

Release 1.4.0

August 23, 2013

CONTENTS

1 Introduction **1**
1.1 Using Futon . . . 1
1.2 Using `curl` . . . 7

2 API Basics **9**
2.1 Request Format and Responses . . . 9
2.2 HTTP Headers . . . 10
2.3 JSON Basics . . . 11
2.4 HTTP Status Codes . . . 12
2.5 HTTP Range Requests . . . 13

3 Configuration **15**
3.1 CouchDB Configuration Files . . . 15
3.2 Configuration File Locations . . . 15
3.3 Update Notifications . . . 15
3.4 MochiWeb Server Options . . . 15
3.5 Socket Options Configuration Setting . . . 15
3.6 Virtual Hosts . . . 16
3.7 HTTP Rewrite Handler . . . 16
3.8 Configuring Server Administrators . . . 17
3.9 OS Daemons . . . 18
3.10 Native SSL Support . . . 18
3.11 HTTP Proxying . . . 19
3.12 Cross-Origin Resource Sharing . . . 20

4 Replication **23**
4.1 Triggering Replication . . . 23
4.2 Replication Procedure . . . 23
4.3 Master - Master replication . . . 23
4.4 Controlling which Documents to Replicate . . . 24
4.5 Migrating Data to Clients . . . 24

5 Replicator Database **25**
5.1 Basics . . . 25
5.2 Documents describing the same replication . . . 26
5.3 Canceling replications . . . 27
5.4 Server restart . . . 27
5.5 Changing the Replicator Database . . . 27
5.6 Replicating the replicator database . . . 28
5.7 Delegations . . . 29

6 Design Docs **31**
6.1 View functions . . . 31
6.2 Show functions . . . 33

6.3 List functions . . . 35
6.4 Update functions . . . 36
6.5 Filter functions . . . 36
6.6 Validate document update functions . . . 38

7 Query Servers **43**
7.1 JavaScript . . . 43
7.2 Erlang . . . 47

8 Changes Feed **49**
8.1 Polling . . . 49
8.2 Long Polling . . . 50
8.3 Continuous . . . 50
8.4 Event Source . . . 51

9 API Reference **53**
9.1 Database Methods . . . 53
9.2 Document Methods . . . 72
9.3 Local (non-replicating) Document Methods . . . 85
9.4 Design Document Methods . . . 87
9.5 Miscellaneous Methods . . . 105
9.6 Configuration Methods . . . 117
9.7 Authentication Methods . . . 121

10 JSON Structure Reference **123**
10.1 All Database Documents . . . 123
10.2 Bulk Document Response . . . 123
10.3 Bulk Documents . . . 123
10.4 Changes information for a database . . . 123
10.5 CouchDB Document . . . 124
10.6 CouchDB Error Status . . . 124
10.7 CouchDB database information object . . . 124
10.8 Design Document . . . 124
10.9 Design Document Information . . . 125
10.10 Document with Attachments . . . 125
10.11 List of Active Tasks . . . 125
10.12 Replication Settings . . . 125
10.13 Replication Status . . . 126
10.14 Request object . . . 126
10.15 Response object . . . 128
10.16 Returned CouchDB Document with Detailed Revision Info . . . 128
10.17 Returned CouchDB Document with Revision Info . . . 128
10.18 Returned Document with Attachments . . . 128
10.19 Security Object . . . 129
10.20 User Context Object . . . 129
10.21 View Head Information . . . 129
10.22 Number Handling . . . 130

11 Configuration Reference **133**
11.1 Configuration Groups . . . 133
11.2 attachments Configuration Options . . . 133
11.3 couchdb Configuration Options . . . 133
11.4 daemons Configuration Options . . . 134
11.5 httpd_db_handlers Configuration Options . . . 134
11.6 couch_httpd_auth Configuration Options . . . 134
11.7 httpd Configuration Options . . . 135
11.8 httpd_design_handlers Configuration Options . . . 135
11.9 httpd_global_handlers Configuration Options . . . 135
11.10 log Configuration Options . . . 135

11.11 query_servers Configuration Options . . . 136
11.12 query_server_config Configuration Options . . . 136
11.13 replicator Configuration Options . . . 136
11.14 stats Configuration Options . . . 136
11.15 uuids Configuration Options . . . 136
11.16 cors Configuration Options . . . 136
11.17 cors vhost Configuration . . . 136

12 Contributing to this Documentation **137**

13 Release History **141**
13.1 1.4.x Branch . . . 141
13.2 1.3.x Branch . . . 142
13.3 1.2.x Branch . . . 146
13.4 1.1.x Branch . . . 149
13.5 1.0.x Branch . . . 151
13.6 0.11.x Branch . . . 156
13.7 0.10.x Branch . . . 160
13.8 0.9.x Branch . . . 161
13.9 0.8.x Branch . . . 164

Index **169**

CHAPTER

ONE

INTRODUCTION

There are two interfaces to CouchDB, the built-in Futon web-based interface and the CouchDB API accessed through the HTTP REST interface. The former is the simplest way to view and monitor your CouchDB installation and perform a number of basic database and system operations. More information on using the Futon interface can be found in *Using Futon*.

The primary way to interact with the CouchDB API is to use a client library or other interface that provides access to the underlying functionality through your chosen language or platform. However, since the API is supported through HTTP REST, you can interact with your CouchDB with any solution that supports the HTTP protocol.

There are a number of different tools that talk the HTTP protocol and allow you to set and configure the necessary information. One tool for this that allows for access from the command-line is `curl`. See *Using curl*.

1.1 Using Futon

Futon is a native web-based interface built into CouchDB. It provides a basic interface to the majority of the functionality, including the ability to create, update, delete and view documents and views, provides access to the configuration parameters, and an interface for initiating replication.

The default view is the Overview page which provides you with a list of the databases. The basic structure of the page is consistent regardless of the section you are in. The main panel on the left provides the main interface to the databases, configuration or replication systems. The side panel on the right provides navigation to the main areas of Futon interface:

The main sections are:

- Overview

  The main overview page, which provides a list of the databases and provides the interface for querying the database and creating and updating documents. See *Managing Databases and Documents*.

- Configuration

  An interface into the configuration of your CouchDB installation. The interface allows you to edit the different configurable parameters. For more details on configuration, see *Configuration*.

- Replicator

  An interface to the replication system, enabling you to initiate replication between local and remote databases. See *Configuring Replication*.

- Status

  Displays a list of the running background tasks on the server. Background tasks include view index building, compaction and replication. The Status page is an interface to the *Active Tasks* API call.

- Verify Installation

  The Verify Installation allows you to check whether all of the components of your CouchDB installation are correctly installed.

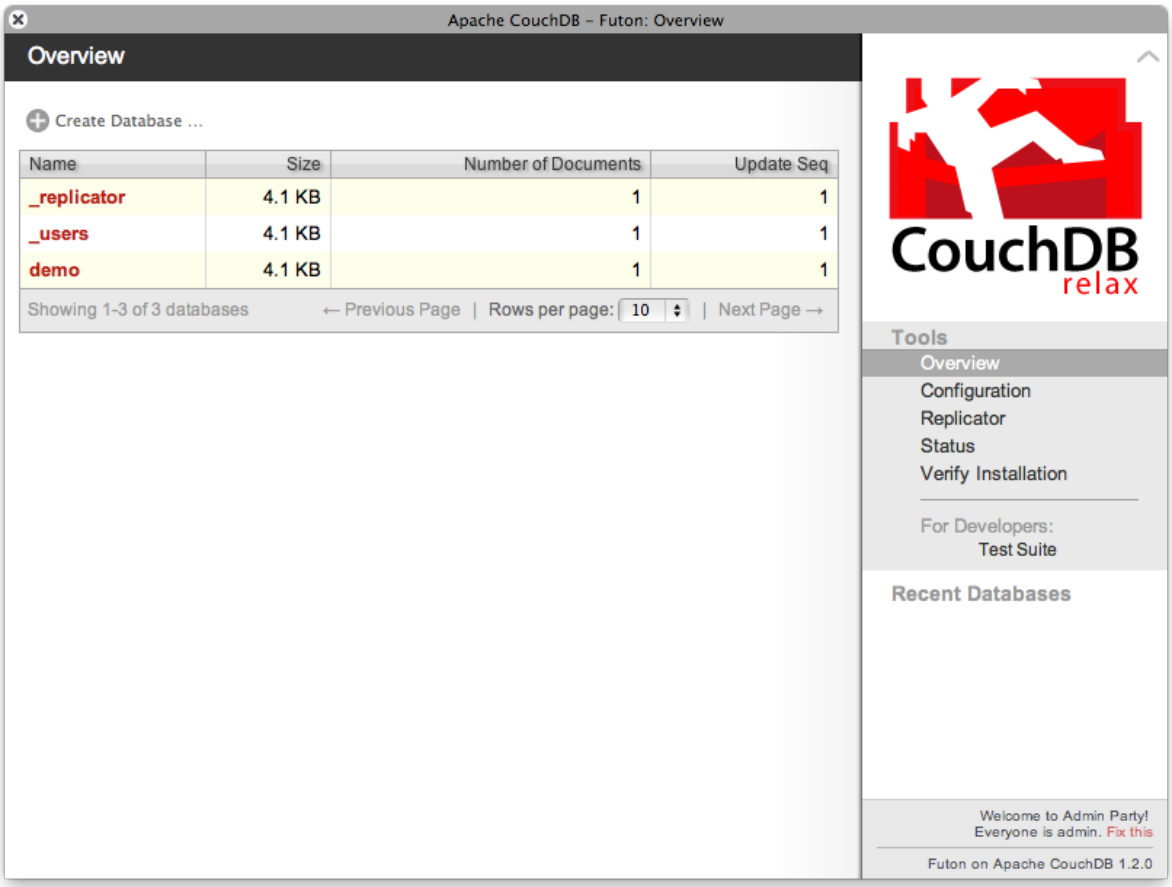

Figure 1.1: Futon Overview

- Test Suite

  The Test Suite section allows you to run the built-in test suite. This executes a number of test routines entirely within your browser to test the API and functionality of your CouchDB installation. If you select this page, you can run the tests by using the Run All button. This will execute all the tests, which may take some time.

1.1.1 Managing Databases and Documents

You can manage databases and documents within Futon using the main Overview section of the Futon interface.

To create a new database, click the Create Database ELLIPSIS button. You will be prompted for the database name, as shown in the figure below.

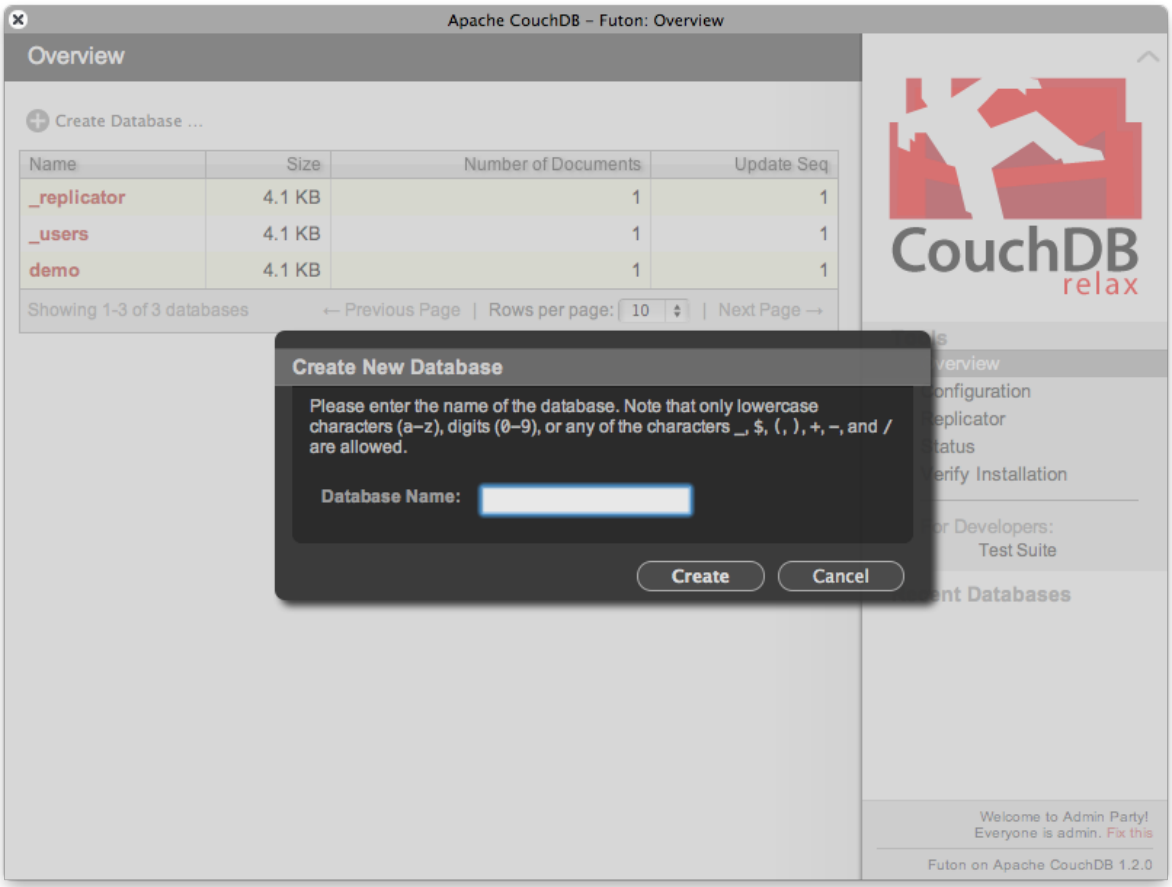

Figure 1.2: Creating a Database

Once you have created the database (or selected an existing one), you will be shown a list of the current documents. If you create a new document, or select an existing document, you will be presented with the edit document display.

Editing documents within Futon requires selecting the document and then editing (and setting) the fields for the document individually before saving the document back into the database.

For example, the figure below shows the editor for a single document, a newly created document with a single ID, the document `_id` field.

To add a field to the document:

1. Click Add Field.
2. In the fieldname box, enter the name of the field you want to create. For example, "company".

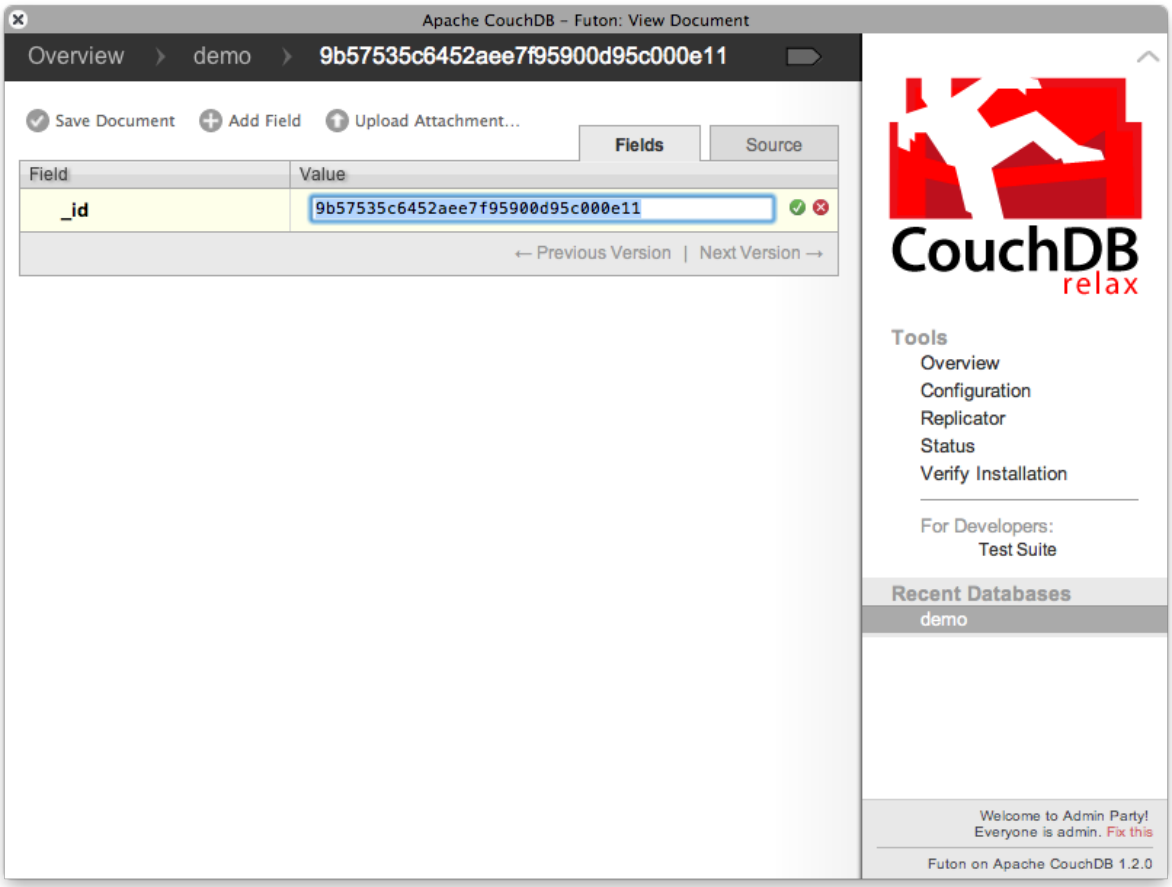

Figure 1.3: Editing a Document

3. Click the green tick next to the field name to confirm the field name change.
4. Double-click the corresponding Value cell.
5. Enter a company name, for example "Example".
6. Click the green tick next to the field value to confirm the field value.
7. The document is still not saved as this point. You must explicitly save the document by clicking the Save Document button at the top of the page. This will save the document, and then display the new document with the saved revision information (the `_rev` field).

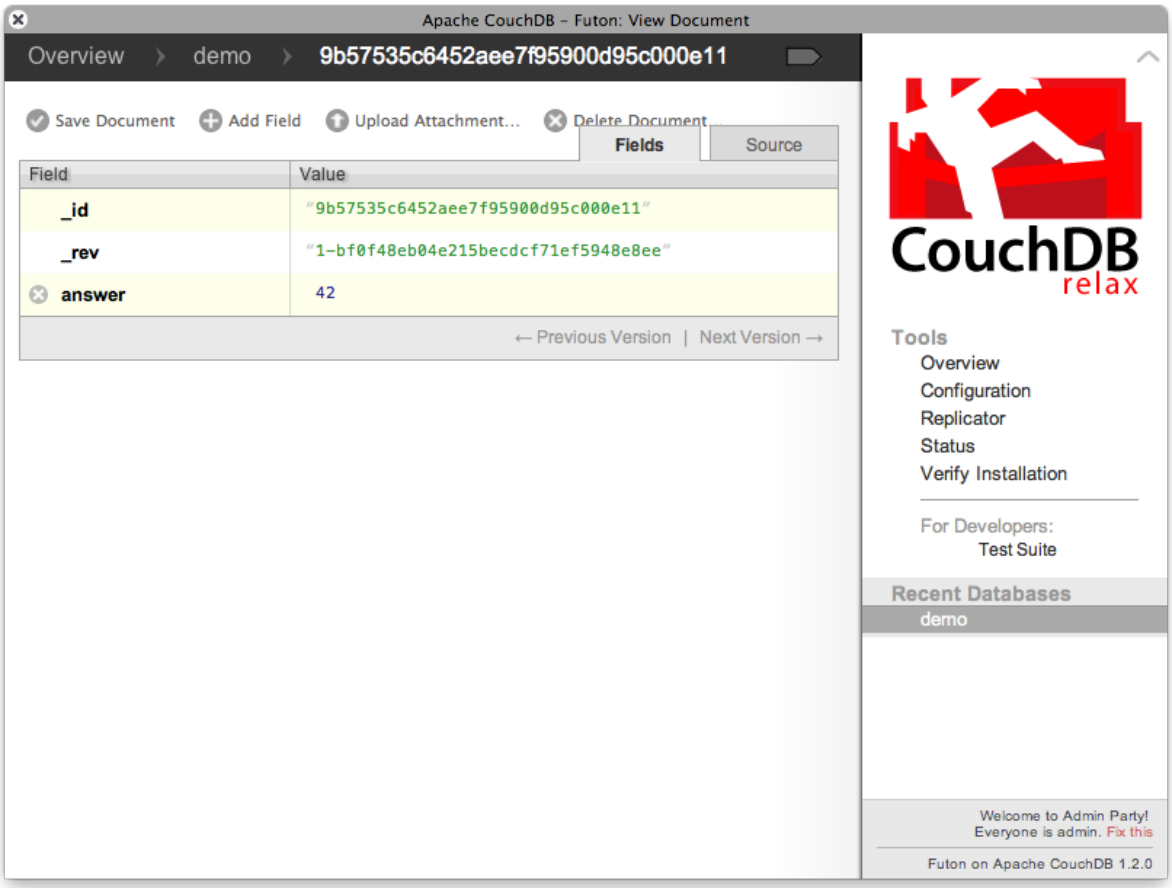

Figure 1.4: Edited Document

The same basic interface is used for all editing operations within Futon. You *must* remember to save the individual element (fieldname, value) using the green tick button, before then saving the document.

1.1.2 Configuring Replication

When you click the Replicator option within the Tools menu you are presented with the Replicator screen. This allows you to start replication between two databases by filling in or select the appropriate options within the form provided.

To start a replication process, either the select the local database or enter a remote database name into the corresponding areas of the form. Replication occurs from the database on the left to the database on the right.

If you are specifying a remote database name, you must specify the full URL of the remote database (including the host, port number and database name). If the remote instance requires authentication, you can specify the username and password as part of the URL, for example `http://username:pass@remotehost:5984/demo`.

To enable continuous replication, click the Continuous checkbox.

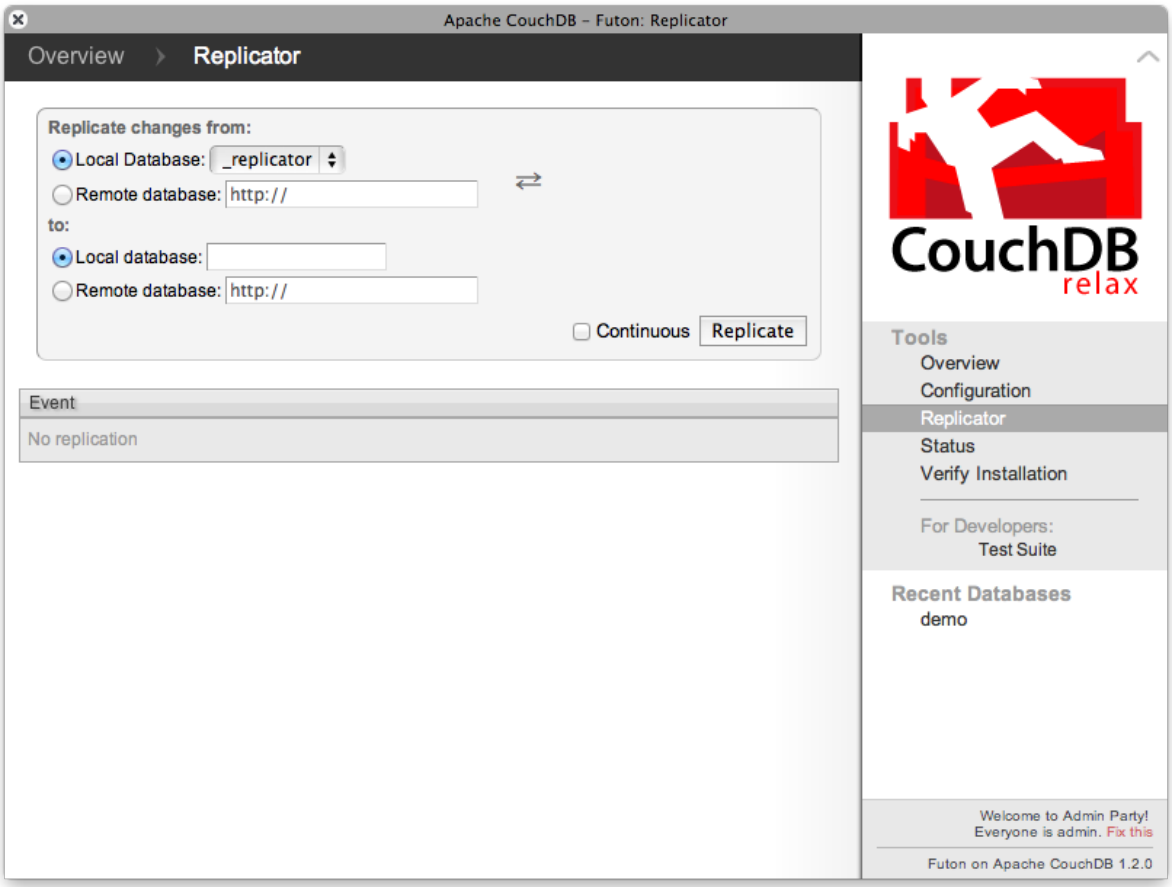

Figure 1.5: Replication Form

To start the replication process, click the Replicate button. The replication process should start and will continue in the background. If the replication process will take a long time, you can monitor the status of the replication using the Status option under the Tools menu.

Once replication has been completed, the page will show the information returned when the replication process completes by the API.

The Replicator tool is an interface to the underlying replication API. For more information, see *POST /_replicate*. For more information on replication, see *Replication*.

1.2 Using `curl`

The `curl` utility is a command line tool available on Unix, Linux, Mac OS X and Windows and many other platforms. `curl` provides easy access to the HTTP protocol (among others) directly from the command-line and is therefore an ideal way of interacting with CouchDB over the HTTP REST API.

For simple `GET` requests you can supply the URL of the request. For example, to get the database information:

```
shell> curl http://127.0.0.1:5984
```

This returns the database information (formatted in the output below for clarity):

```
{
   "couchdb" : "Welcome",
   "version" : "|version|",
}
```

Note: For some URLs, especially those that include special characters such as ampersand, exclamation mark, or question mark, you should quote the URL you are specifying on the command line. For example:

```
shell> curl 'http://couchdb:5984/_uuids?count=5'
```

You can explicitly set the HTTP command using the `-X` command line option. For example, when creating a database, you set the name of the database in the URL you send using a PUT request:

```
shell> curl -X PUT http://127.0.0.1:5984/demo
{"ok":true}
```

But to obtain the database information you use a `GET` request (with the return information formatted for clarity):

```
shell> curl -X GET http://127.0.0.1:5984/demo
{
   "compact_running" : false,
   "doc_count" : 0,
   "db_name" : "demo",
   "purge_seq" : 0,
   "committed_update_seq" : 0,
   "doc_del_count" : 0,
   "disk_format_version" : 5,
   "update_seq" : 0,
   "instance_start_time" : "1306421773496000",
   "disk_size" : 79
}
```

For certain operations, you must specify the content type of request, which you do by specifying the `Content-Type` header using the `-H` command-line option:

```
shell> curl -H 'Content-Type: application/json' http://127.0.0.1:5984/_uuids
```

You can also submit 'payload' data, that is, data in the body of the HTTP request using the `-d` option. This is useful if you need to submit JSON structures, for example document data, as part of the request. For example, to submit a simple document to the `demo` database:

```
shell> curl -H 'Content-Type: application/json' \
            -X POST http://127.0.0.1:5984/demo \
            -d '{"company": "Example, Inc."}'
{"ok":true,"id":"8843faaf0b831d364278331bc3001bd8",
 "rev":"1-33b9fbce46930280dab37d672bbc8bb9"}
```

In the above example, the argument after the `-d` option is the JSON of the document we want to submit.

The document can be accessed by using the automatically generated document ID that was returned:

```
shell> curl -X GET http://127.0.0.1:5984/demo/8843faaf0b831d364278331bc3001bd8
{"_id":"8843faaf0b831d364278331bc3001bd8",
 "_rev":"1-33b9fbce46930280dab37d672bbc8bb9",
 "company":"Example, Inc."}
```

The API samples in the *API Basics* show the HTTP command, URL and any payload information that needs to be submitted (and the expected return value). All of these examples can be reproduced using `curl` with the command-line examples shown above.

CHAPTER

TWO

API BASICS

The CouchDB API is the primary method of interfacing to a CouchDB instance. Requests are made using HTTP and requests are used to request information from the database, store new data, and perform views and formatting of the information stored within the documents.

Requests to the API can be categorised by the different areas of the CouchDB system that you are accessing, and the HTTP method used to send the request. Different methods imply different operations, for example retrieval of information from the database is typically handled by the `GET` operation, while updates are handled by either a `POST` or `PUT` request. There are some differences between the information that must be supplied for the different methods. For a guide to the basic HTTP methods and request structure, see *Request Format and Responses*.

For nearly all operations, the submitted data, and the returned data structure, is defined within a JavaScript Object Notation (JSON) object. Basic information on the content and data types for JSON are provided in *JSON Basics*.

Errors when accessing the CouchDB API are reported using standard HTTP Status Codes. A guide to the generic codes returned by CouchDB are provided in *HTTP Status Codes*.

When accessing specific areas of the CouchDB API, specific information and examples on the HTTP methods and request, JSON structures, and error codes are provided. For a guide to the different areas of the API, see *API Reference*.

2.1 Request Format and Responses

CouchDB supports the following HTTP request methods:

- `GET`

  Request the specified item. As with normal HTTP requests, the format of the URL defines what is returned. With CouchDB this can include static items, database documents, and configuration and statistical information. In most cases the information is returned in the form of a JSON document.

- `HEAD`

  The `HEAD` method is used to get the HTTP header of a `GET` request without the body of the response.

- `POST`

  Upload data. Within CouchDB `POST` is used to set values, including uploading documents, setting document values, and starting certain administration commands.

- `PUT`

  Used to put a specified resource. In CouchDB `PUT` is used to create new objects, including databases, documents, views and design documents.

- `DELETE`

  Deletes the specified resource, including documents, views, and design documents.

- `COPY`

  A special method that can be used to copy documents and objects.

If you use the an unsupported HTTP request type with a URL that does not support the specified type, a 405 error will be returned, listing the supported HTTP methods. For example:

```
{
    "error":"method_not_allowed",
    "reason":"Only GET,HEAD allowed"
}
```

The CouchDB design document API and the functions when returning HTML (for example as part of a show or list) enables you to include custom HTTP headers through the `headers` block of the return object.

2.2 HTTP Headers

Because CouchDB uses HTTP for all communication, you need to ensure that the correct HTTP headers are supplied (and processed on retrieval) so that you get the right format and encoding. Different environments and clients will be more or less strict on the effect of these HTTP headers (especially when not present). Where possible you should be as specific as possible.

2.2.1 Request Headers

- `Content-type`

  Specifies the content type of the information being supplied within the request. The specification uses MIME type specifications. For the majority of requests this will be JSON (`application/json`). For some settings the MIME type will be plain text. When uploading attachments it should be the corresponding MIME type for the attachment or binary (`application/octet-stream`).

  The use of the `Content-type` on a request is highly recommended.

- `Accept`

  Specifies the list of accepted data types to be returned by the server (i.e. that are accepted/understandable by the client). The format should be a list of one or more MIME types, separated by colons.

  For the majority of requests the definition should be for JSON data (`application/json`). For attachments you can either specify the MIME type explicitly, or use `*/*` to specify that all file types are supported. If the `Accept` header is not supplied, then the `*/*` MIME type is assumed (i.e. client accepts all formats).

  The use of `Accept` in queries for CouchDB is not required, but is highly recommended as it helps to ensure that the data returned can be processed by the client.

  If you specify a data type using the `Accept` header, CouchDB will honor the specified type in the `Content-type` header field returned. For example, if you explicitly request `application/json` in the `Accept` of a request, the returned HTTP headers will use the value in the returned `Content-type` field.

  For example, when sending a request without an explicit `Accept` header, or when specifying `*/*`:

  ```
  GET /recipes HTTP/1.1
  Host: couchdb:5984
  Accept: */*
  ```

  The returned headers are:

  ```
  Server: CouchDB/1.0.1 (Erlang OTP/R13B)
  Date: Thu, 13 Jan 2011 13:39:34 GMT
  Content-Type: text/plain;charset=utf-8
  Content-Length: 227
  Cache-Control: must-revalidate
  ```

  Note that the returned content type is `text/plain` even though the information returned by the request is in JSON format.

Explicitly specifying the `Accept` header:

```
GET /recipes HTTP/1.1
Host: couchdb:5984
Accept: application/json
```

The headers returned include the `application/json` content type:

```
Server: CouchDB/|version| (Erlang OTP/R13B)
Date: Thu, 13 Jan 2011 13:40:11 GMT
Content-Type: application/json
Content-Length: 227
Cache-Control: must-revalidate
```

2.2.2 Response Headers

Response headers are returned by the server when sending back content and include a number of different header fields, many of which are standard HTTP response header and have no significance to CouchDB operation. The list of response headers important to CouchDB are listed below.

- `Content-type`

  Specifies the MIME type of the returned data. For most request, the returned MIME type is `text/plain`. All text is encoded in Unicode (UTF-8), and this is explicitly stated in the returned `Content-type`, as `text/plain;charset=utf-8`.

- `Cache-control`

  The cache control HTTP response header provides a suggestion for client caching mechanisms on how to treat the returned information. CouchDB typically returns the `must-revalidate`, which indicates that the information should be revalidated if possible. This is used to ensure that the dynamic nature of the content is correctly updated.

- `Content-length`

  The length (in bytes) of the returned content.

- `Etag`

  The `Etag` HTTP header field is used to show the revision for a document, or a view.

  ETags have been assigned to a map/reduce group (the collection of views in a single design document). Any change to any of the indexes for those views would generate a new ETag for all view URL's in a single design doc, even if that specific view's results had not changed.

  Each `_view` URL has its own ETag which only gets updated when changes are made to the database that effect that index. If the index for that specific view does not change, that view keeps the original ETag head (therefore sending back 304 Not Modified more often).

2.3 JSON Basics

The majority of requests and responses to CouchDB use the JavaScript Object Notation (JSON) for formatting the content and structure of the data and responses.

JSON is used because it is the simplest and easiest to use solution for working with data within a web browser, as JSON structures can be evaluated and used as JavaScript objects within the web browser environment. JSON also integrates with the server-side JavaScript used within CouchDB.

JSON supports the same basic types as supported by JavaScript, these are:

- Number (either integer or floating-point).
- String; this should be enclosed by double-quotes and supports Unicode characters and backslash escaping. For example:

```
"A String"
```

- Boolean - a `true` or `false` value. You can use these strings directly. For example:

```
{ "value": true}
```

- Array - a list of values enclosed in square brackets. For example:

```
["one", "two", "three"]
```

- Object - a set of key/value pairs (i.e. an associative array, or hash). The key must be a string, but the value can be any of the supported JSON values. For example:

```
{
   "servings" : 4,
   "subtitle" : "Easy to make in advance, and then cook when ready",
   "cooktime" : 60,
   "title" : "Chicken Coriander"
}
```

  In CouchDB, the JSON object is used to represent a variety of structures, including the main CouchDB document.

Parsing JSON into a JavaScript object is supported through the `JSON.parse()` function in JavaScript, or through various libraries that will perform the parsing of the content into a JavaScript object for you. Libraries for parsing and generating JSON are available in many languages, including Perl, Python, Ruby, Erlang and others.

> **Warning:** Care should be taken to ensure that your JSON structures are valid, invalid structures will cause CouchDB to return an HTTP status code of 500 (server error).

2.4 HTTP Status Codes

With the interface to CouchDB working through HTTP, error codes and statuses are reported using a combination of the HTTP status code number, and corresponding data in the body of the response data.

A list of the error codes returned by CouchDB, and generic descriptions of the related errors are provided below. The meaning of different status codes for specific request types are provided in the corresponding API call reference.

- `200 - OK`

  Request completed successfully.

- `201 - Created`

  Document created successfully.

- `202 - Accepted`

  Request has been accepted, but the corresponding operation may not have completed. This is used for background operations, such as database compaction.

- `304 - Not Modified`

  The additional content requested has not been modified. This is used with the ETag system to identify the version of information returned.

- `400 - Bad Request`

  Bad request structure. The error can indicate an error with the request URL, path or headers. Differences in the supplied MD5 hash and content also trigger this error, as this may indicate message corruption.

- `401 - Unauthorized`

  The item requested was not available using the supplied authorization, or authorization was not supplied.

- `403 - Forbidden`

  The requested item or operation is forbidden.

- `404 - Not Found`

  The requested content could not be found. The content will include further information, as a JSON object, if available. The structure will contain two keys, `error` and `reason`. For example:

  ```
  {"error":"not_found","reason":"no_db_file"}
  ```

- `405 - Resource Not Allowed`

  A request was made using an invalid HTTP request type for the URL requested. For example, you have requested a `PUT` when a `POST` is required. Errors of this type can also triggered by invalid URL strings.

- `406 - Not Acceptable`

  The requested content type is not supported by the server.

- `409 - Conflict`

  Request resulted in an update conflict.

- `412 - Precondition Failed`

  The request headers from the client and the capabilities of the server do not match.

- `415 - Bad Content Type`

  The content types supported, and the content type of the information being requested or submitted indicate that the content type is not supported.

- `416 - Requested Range Not Satisfiable`

  The range specified in the request header cannot be satisfied by the server.

- `417 - Expectation Failed`

  When sending documents in bulk, the bulk load operation failed.

- `500 - Internal Server Error`

  The request was invalid, either because the supplied JSON was invalid, or invalid information was supplied as part of the request.

2.5 HTTP Range Requests

HTTP allows you to specify byte ranges for requests. This allows the implementation of resumable downloads and skippable audio and video streams alike. This is available for all attachments inside CouchDB.

This is just a real quick run through how this looks under the hood. Usually, you will have larger binary files to serve from CouchDB, like MP3s and videos, but to make things a little more obvious, I use a text file here (Note that I use the `application/octet-stream` Content-Type instead of `text/plain`).

```
shell> cat file.txt
My hovercraft is full of eels!
```

Now let's store this text file as an attachment in CouchDB. First, we create a database:

```
shell> curl -X PUT http://127.0.0.1:5984/test
{"ok":true}
```

Then we create a new document and the file attachment in one go:

```
shell> curl -X PUT http://127.0.0.1:5984/test/doc/file.txt \
            -H "Content-Type: application/octet-stream" -d@file.txt
{"ok":true,"id":"doc","rev":"1-287a28fa680ae0c7fb4729bf0c6e0cf2"}
```

Now we can request the whole file easily:

```
shell> curl -X GET http://127.0.0.1:5984/test/doc/file.txt
My hovercraft is full of eels!
```

But say we only want the first 13 bytes:

```
shell> curl -X GET http://127.0.0.1:5984/test/doc/file.txt \
            -H "Range: bytes=0-12"
My hovercraft
```

HTTP supports many ways to specify single and even multiple byte ranges. Read all about it in RFC 2616.

Note: Databases that have been created with CouchDB 1.0.2 or earlier will support range requests in 1.4, but they are using a less-optimal algorithm. If you plan to make heavy use of this feature, make sure to compact your database with CouchDB 1.4 to take advantage of a better algorithm to find byte ranges.

CHAPTER

THREE

CONFIGURATION

3.1 CouchDB Configuration Files

3.2 Configuration File Locations

CouchDB reads files from the following locations, in the following order.

1. `PREFIX/default.ini`
2. `PREFIX/default.d/*`
3. `PREFIX/local.ini`
4. `PREFIX/local.d/*`

Settings in successive documents override the settings in earlier entries. For example, setting the `bind_address` parameter in `local.ini` would override any setting in `default.ini`.

> **Warning:** The `default.ini` file may be overwritten during an upgrade or re-installation, so localised changes should be made to the `local.ini` file or files within the `local.d` directory.

3.3 Update Notifications

3.4 MochiWeb Server Options

Server options for the MochiWeb component of CouchDB can be added to the configuration files. Settings should be added to the `server_options` option of the `[httpd]` section of `local.ini`. For example:

```
[httpd]
server_options = [{backlog, 128}, {acceptor_pool_size, 16}]
```

3.5 Socket Options Configuration Setting

The socket options for the listening socket in CouchDB can now be set within the CouchDB configuration file. The setting should be added to the `[httpd]` section of the file using the option name `socket_options`. The specification is as a list of tuples. For example:

```
[httpd]
socket_options = [{recbuf, 262144}, {sndbuf, 262144}, {nodelay, true}]
```

The options supported are a subset of full options supported by the TCP/IP stack. A list of the supported options are provided in the Erlang inet documentation.

3.6 Virtual Hosts

CouchDB, since 0.11.0, can map requests to different locations based on the `Host` header, even if they arrive on the some inbound IP address.

This allows different virtual hosts on the same machine to map to different databases or design documents, etc. The most common use case is to map a virtual host to a Rewrite Handler, to provide full control over the application's URIs.

To add a virtual host, add a CNAME pointer to the DNS for your domain name. For development and testing, it is sufficient to add an entry in the hosts file, typically */etc/hosts*' on Unix-like operating systems:

```
# CouchDB vhost definitions, refer to local.ini for further details
127.0.0.1       sofa.couchdb
```

Test that this is working:

```
$ ping sofa.couchdb
PING sofa.couchdb (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost.localdomain (127.0.0.1): icmp_req=1 ttl=64 time=0.025 ms
64 bytes from localhost.localdomain (127.0.0.1): icmp_req=2 ttl=64 time=0.051 ms
^C
```

Finally, add an entry to your *configuration file* in the `[vhosts]` section:

```
[vhosts]
sofa.couchdb:5984 = /sofa/_design/sofa/_rewrite
```

If your CouchDB is listening on the default HTTP port, or is sitting behind a proxy, then don't specify a port number in the vhost key.

With the above setup, a request to `http://sofa.couchdb:5984/sweet-o` will be mapped to `http://127.0.0.1:5984/sofa/_design/sofa/_rewrite/sweet-o` New in version 0.11.0: added *vhosts* functionality

3.7 HTTP Rewrite Handler

Following on from virtual hosts, CouchDB includes a custom URL rewriter. All rewriting is done from `/dbname/_design/ddocname/_rewrite` by default.

The rewriter is flexible, and can handle methods and custom query formats.

Each rule should be in the `rewrites` top-level key of the design doc. Example of a complete rule :

```
{
    ....
    "rewrites": [
    {
        "from": "",
        "to": "index.html",
        "method": "GET",
        "query": {}
    }
    ]
}
```

from: is the path rule used to bind current uri to the rule. It uses pattern matching for that.

to: rule to rewrite an url. It can contain variables depending on binding variables discovered during pattern matching and query args (url args and from the query member.)

method: method to bind the request method to the rule. If method is missing, any method will be matched in the rewrite.

query: optional query arguments, that may contain dynamic variables, by binding keys in the to be used with the matching URL.

`to` and `from` are paths with patterns. The pattern can be strings starting with `:` or `*`, for example `/somepath/:var/*`.

The pattern matching is done by first matching the request method to a rule. Then it will try to match the path to one specific rule. If no rule match, then a 404 error is displayed.

The path is converted into an erlang list, by regex splitting on `/`. Each variable is converted into an atom. The subsequent pattern matching step is done by splitting `/` in the request url into a list of atoms. A string pattern will match the equivalent token. The `*` atom will match any number of tokens, but may only be present as the last pattern in the path. If all tokens are matched, and all path terms have been consumed, then the overall path specification matches.

Once a matching `from` rule is found we rewrite the request url using the `from`, `to`, and `query` members. Each identified token will be reused within the rule, and in the subsequent query if required. The identified tokens are matched to the rule and will replace var. If `*` is found in the rule it will contain any remaining suffix.

The rewriter is re-entrant, and has a configurable recursion limit, set by default at 100.

3.8 Configuring Server Administrators

A default CouchDB install provides admin-level access to all connecting users. This configuration is known as `Admin Party`, and is not recommended for in-production usage. You can crash the party simply by creating the first admin account. CouchDB server administrators and passwords are not stored in the `_users` database, but in the `local.ini` file, which should be appropriately secured and readable only by system administrators.

```
[admins]
;admin = mysecretpassword
admin = -hashed-6d3c30241ba0aaa4e16c6ea99224f915687ed8cd,7f4a3e05e0cbc6f48a0035e3508eef90
architect = -pbkdf2-43ecbd256a70a3a2f7de40d2374b6c3002918834,921a12f74df0c1052b3e562a23cd227f,100
```

Administrators can be added directly to the `[admins]` section, and when CouchDB is restarted, the passwords will be salted and encrypted. You may also use the HTTP interface to create administrator accounts; this way, you don't need to restart CouchDB, and there's no need to temporarily store or transmit passwords in plaintext. The HTTP `_config/admins` endpoint supports querying, deleting or creating new admin accounts:

```
shell> GET /_config/admins HTTP/1.1
    Accept: application/json
    Host: localhost:5984

HTTP/1.1 200 OK
    Cache-Control: must-revalidate
    Content-Length: 196
    Content-Type: application/json
    Date: Fri, 30 Nov 2012 11:37:18 GMT
    Server: CouchDB/1.3.0 (Erlang OTP/R15B02)

{
    "admin": "-hashed-6d3c30241ba0aaa4e16c6ea99224f915687ed8cd,7f4a3e05e0cbc6f48a0035e3508eef90",
    "architect": "-pbkdf2-43ecbd256a70a3a2f7de40d2374b6c3002918834,921a12f74df0c1052b3e562a23cd22
}
```

If you already have a salted, encrypted password string (for example, from an old `local.ini` file, or from a different CouchDB server), then you can store the "raw" encrypted string, without having CouchDB doubly encrypt it.

```
shell> PUT /_config/admins/architect?raw=true HTTP/1.1
    Accept: application/json
    Content-Type: application/json
    Content-Length: 89
```

```
    Host: localhost:5984

    "-pbkdf2-43ecbd256a70a3a2f7de40d2374b6c3002918834,921a12f74df0c1052b3e562a23cd227f,10000"

HTTP/1.1 200 OK
    Cache-Control: must-revalidate
    Content-Length: 89
    Content-Type: application/json
    Date: Fri, 30 Nov 2012 11:39:18 GMT
    Server: CouchDB/1.3.0 (Erlang OTP/R15B02)

"-pbkdf2-43ecbd256a70a3a2f7de40d2374b6c3002918834,921a12f74df0c1052b3e562a23cd227f,10000"
```

Further details are available in `security_`, including configuring the work factor for `PBKDF2`, and the algorithm itself at PBKDF2 (RFC-2898). New in version 1.3.0: `PBKDF2` server-side hashed salted password support added, now as a synchronous call for the `_config/admins` API.

3.9 OS Daemons

CouchDB now supports starting external processes. The support is simple and enables CouchDB to start each configured OS daemon. If the daemon stops at any point, CouchDB will restart it (with protection to ensure regularly failing daemons are not repeatedly restarted).

The daemon starting process is one-to-one; for each each configured daemon in the configuration file, CouchDB will start exactly one instance. If you need to run multiple instances, then you must create separate individual configurations. Daemons are configured within the `[os_daemons]` section of your configuration file (`local.ini`). The format of each configured daemon is:

```
NAME = PATH ARGS
```

Where `NAME` is an arbitrary (and unique) name to identify the daemon; `PATH` is the full path to the daemon to be executed; `ARGS` are any required arguments to the daemon.

For example:

```
[os_daemons]
basic_responder = /usr/local/bin/responder.js
```

There is no interactivity between CouchDB and the running process, but you can use the OS Daemons service to create new HTTP servers and responders and then use the new proxy service to redirect requests and output to the CouchDB managed service. For more information on proxying, see *HTTP Proxying*. For further background on the OS Daemon service, see CouchDB Externals API.

3.10 Native SSL Support

CouchDB 1.4 supports SSL natively. All your secure connection needs can now be served without needing to setup and maintain a separate proxy server that handles SSL.

SSL setup can be tricky, but the configuration in CouchDB was designed to be as easy as possible. All you need is two files; a certificate and a private key. If you bought an official SSL certificate from a certificate authority, both should be in your possession already.

If you just want to try this out and don't want to pay anything upfront, you can create a self-signed certificate. Everything will work the same, but clients will get a warning about an insecure certificate.

You will need the OpenSSL command line tool installed. It probably already is.

```
shell> mkdir cert && cd cert
shell> openssl genrsa > privkey.pem
shell> openssl req -new -x509 -key privkey.pem -out mycert.pem -days 1095
```

```
shell> ls
mycert.pem privkey.pem
```

Now, you need to edit CouchDB's configuration, either by editing your `local.ini` file or using the `/_config` API calls or the configuration screen in Futon. Here is what you need to do in `local.ini`, you can infer what needs doing in the other places.

Be sure to make these edits. Under `[daemons]` you should see:

```
; enable SSL support by uncommenting the following line and supply the PEM's below.
; the default ssl port CouchDB listens on is 6984
;httpsd = {couch_httpd, start_link, [https]}
```

Here uncomment the last line:

```
httpsd = {couch_httpd, start_link, [https]}
```

Next, under `[ssl]` you will see:

```
;cert_file = /full/path/to/server_cert.pem
;key_file = /full/path/to/server_key.pem
```

Uncomment and adjust the paths so it matches your system's paths:

```
cert_file = /home/jan/cert/mycert.pem
key_file = /home/jan/cert/privkey.pem
```

For more information please read http://www.openssl.org/docs/HOWTO/certificates.txt.

Now start (or restart) CouchDB. You should be able to connect to it using HTTPS on port 6984:

```
shell> curl https://127.0.0.1:6984/
curl: (60) SSL certificate problem, verify that the CA cert is OK. Details:
error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed
More details here: http://curl.haxx.se/docs/sslcerts.html

curl performs SSL certificate verification by default, using a "bundle"
of Certificate Authority (CA) public keys (CA certs). If the default
bundle file isn't adequate, you can specify an alternate file
using the --cacert option.
If this HTTPS server uses a certificate signed by a CA represented in
the bundle, the certificate verification probably failed due to a
problem with the certificate (it might be expired, or the name might
not match the domain name in the URL).
If you'd like to turn off curl's verification of the certificate, use
the -k (or --insecure) option.
```

Oh no what happened?! — Remember, clients will notify their users that your certificate is self signed. `curl` is the client in this case and it notifies you. Luckily you trust yourself (don't you?) and you can specify the `-k` option as the message reads:

```
shell> curl -k https://127.0.0.1:6984/
{"couchdb":"Welcome","version":"|version|"}
```

All done.

3.11 HTTP Proxying

The HTTP proxy feature makes it easy to map and redirect different content through your CouchDB URL. The proxy works by mapping a pathname and passing all content after that prefix through to the configured proxy address.

Configuration of the proxy redirect is handled through the `[httpd_global_handlers]` section of the CouchDB configuration file (typically `local.ini`). The format is:

```
[httpd_global_handlers]
PREFIX = {couch_httpd_proxy, handle_proxy_req, <<"DESTINATION">>}
```

Where:

- `PREFIX`

  Is the string that will be matched. The string can be any valid qualifier, although to ensure that existing database names are not overridden by a proxy configuration, you can use an underscore prefix.

- `DESTINATION`

  The fully-qualified URL to which the request should be sent. The destination must include the `http` prefix. The content is used verbatim in the original request, so you can also forward to servers on different ports and to specific paths on the target host.

The proxy process then translates requests of the form:

```
http://couchdb:5984/PREFIX/path
```

To:

```
DESTINATION/path
```

Note: Everything after `PREFIX` including the required forward slash will be appended to the `DESTINATION`.

The response is then communicated back to the original client.

For example, the following configuration:

```
_google = {couch_httpd_proxy, handle_proxy_req, <<"http://www.google.com">>}
```

Would forward all requests for `http://couchdb:5984/_google` to the Google website.

The service can also be used to forward to related CouchDB services, such as Lucene:

```
[httpd_global_handlers]
_fti = {couch_httpd_proxy, handle_proxy_req, <<"http://127.0.0.1:5985">>}
```

Note: The proxy service is basic. If the request is not identified by the `DESTINATION`, or the remainder of the `PATH` specification is incomplete, the original request URL is interpreted as if the `PREFIX` component of that URL does not exist.

For example, requesting `http://couchdb:5984/_intranet/media` when `/media` on the proxy destination does not exist, will cause the request URL to be interpreted as `http://couchdb:5984/media`. Care should be taken to ensure that both requested URLs and destination URLs are able to cope.

3.12 Cross-Origin Resource Sharing

CORS, or "Cross-Origin Resource Sharing", allows a resource such as a web page running JavaScript inside a browser, to make AJAX requests (XMLHttpRequests) to a different domain, without compromising the security of either party.

A typical use case is to have a static website hosted on a CDN make requests to another resource, such as a hosted CouchDB instance. This avoids needing an intermediary proxy, using JSONP or similar workarounds to retrieve and host content.

While CouchDB's integrated HTTP server and support for document attachments makes this less of a constraint for pure CouchDB projects, there are many cases where separating the static content from the database access is desirable, and CORS makes this very straightforward.

By supporting CORS functionality, a CouchDB instance can accept direct connections to protected databases and instances, without the browser functionality being blocked due to same-origin constraints. CORS is supported today on over 90% of recent browsers.

CORS support is provided as experimental functionality in 1.3.0, and as such will need to be enabled specifically in CouchDB's configuration. While all origins are forbidden from making requests by default, support is available for simple requests, preflight requests and per-vhost configuration. New in version 1.3.0.

3.12.1 Enabling CORS

To enable CORS support, you need to set the `enable_cors = true` option in the `[httpd]` section of `local.ini`, and add a `[cors]` section containing a `origins = *` setting. Note that by default, no origins are accepted; you must either use a wildcard or whitelist.

```
[httpd]
enable_cors = true

[cors]
origins = *
```

3.12.2 Passing Credentials

By default, neither authentication headers nor cookies are included in requests and responses. To do so requires both setting *XmlHttpRequest.withCredentials = true* on the request object in the browser and enabling credentials support in CouchDB.

```
[cors]
credentials = true
```

CouchDB will respond to a credentials-enabled CORS request with an additional header, *Access-Control-Allow-Credentials=true*.

3.12.3 Tightening Access

Access can be restricted by protocol, host and optionally by port:

```
[cors]
; List of origins, separated by a comma (protocol, host, optional port)
; refer to http://tools.ietf.org/html/rfc6454 for specification
origins = http://localhost, https://localhost, http://www.number10.gov.uk:80
```

Specific HTTP methods may also be restricted:

```
[cors]
; List of accepted methods, comma-separated
; refer to http://tools.ietf.org/html/rfc2616, rfc2817, rfc5789
methods = GET, POST, PUT, DELETE
```

You can allow additional HTTP header fields to be allowed:

```
[cors]
; List of accepted headers separated by a comma
headers = Authorization, Cookie
```

Note that *Authorization* and *Cookie* are not part of the standard set.

3.12.4 Configuration per vhost

All CORS-related settings may be configured on a per-vhost basis. For example, the configuration section for *http://example.com/* would be contained in:

```
[cors:http://example.com]
credentials = false
origins = *
methods = GET, PUT, HEAD
```

3.12.5 Useful References

- Original JIRA implementation ticket

Standards and References:

- IETF RFCs relating to methods RFC 2618, RFC 2817, and RFC 5789
- IETF RFC 6454 for Web Origins
- W3C CORS standard

Mozilla Developer Network Resources:

- Same origin policy for URIs
- HTTP Access Control
- Server-side Access Control
- Javascript same origin policy

Client-side CORS support and usage:

- CORS browser support matrix
- CORS tutorial
- Cross-Site XMLHttpRequests with CORS

CHAPTER

FOUR

REPLICATION

One of CouchDB's strengths is the ability to synchronize two copies of the same database. This enables users to distribute data across several nodes or datacenters, but also to move data more closely to clients.

Replication involves a source and a destination database, which can be one the same or on different CouchDB instances. 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 on the destination database (if they even existed).

4.1 Triggering Replication

Replication is controlled through documents in the *Replicator Database*, where each document describes one replication process (see *Replication Settings*).

A replication is triggered by storing a replication document in the replicator database. Its status can be inspected through the active tasks API (see *GET /_active_tasks* and *Replication Status*). A replication can be stopped by deleting the document, or by updating it with its *cancel* property set to *true*.

4.2 Replication Procedure

During replication, CouchDB will compare the source and the destination database to determine which documents differ between the source and the destination database. It does so by following the *Changes Feed* on the source and comparing the documents to the destination. Changes are submitted to the destination in batches where they can introduce conflicts. Documents that already exist on the destination in the same revision are not transferred. As the deletion of documents is represented by a new revision, a document deleted on the source will also be deleted on the target.

A replication task will finish once it reaches the end of the changes feed. If its *continuous* property is set to true, it will wait for new changes to appear until the task is cancelled. Replication tasks also create checkpoint documents on the destination to ensure that a restarted task can continue from where it stopped, for example after it has crashed.

When a replication task is initiated on the sending node, it is called *push* replication, if it is initiated by the receiving node, it is called *pull* replication.

4.3 Master - Master replication

One replication task will only transfer changes in one direction. To achieve master-master replication it is possible to set up two replication tasks in different directions. When a change is replication from database A to B by the first task, the second will discover that the new change on B already exists in A and will wait for further changes.

4.4 Controlling which Documents to Replicate

There are two ways for controlling which documents are replicated, and which are skipped. *Local* documents are never replicated (see *Local (non-replicating) Document Methods*).

Additionally, *Filter functions* can be used in a replication documents (see *Replication Settings*). The replication task will then evaluate the filter function for each document in the changes feed. The document will only be replicated if the filter returns *true*.

4.5 Migrating Data to Clients

Replication can be especially useful for bringing data closer to clients. PouchDB implements the replication algorithm of CouchDB in JavaScript, making it possible to make data from a CouchDB database available in an offline browser application, and synchronize changes back to CouchDB.

CHAPTER

FIVE

REPLICATOR DATABASE

A database where you `PUT`/`POST` documents to trigger replications and you `DELETE` to cancel ongoing replications. These documents have exactly the same content as the JSON objects we used to `POST` to `_replicate` (fields `source`, `target`, `create_target`, `continuous`, `doc_ids`, `filter`, `query_params`.

Replication documents can have a user defined `_id`. Design documents (and `_local` documents) added to the replicator database are ignored.

The default name of this database is `_replicator`. The name can be changed in the `local.ini` configuration, section `[replicator]`, parameter `db`.

5.1 Basics

Let's say you PUT the following document into `_replicator`:

```
{
    "_id": "my_rep",
    "source":  "http://myserver.com:5984/foo",
    "target":  "bar",
    "create_target":  true
}
```

In the couch log you'll see 2 entries like these:

```
[Thu, 17 Feb 2011 19:43:59 GMT] [info] [<0.291.0>] Document `my_rep` triggered replication `c0ebe
[Thu, 17 Feb 2011 19:44:37 GMT] [info] [<0.124.0>] Replication `c0ebe9256695ff083347cbf95f93e280+
```

As soon as the replication is triggered, the document will be updated by CouchDB with 3 new fields:

```
{
    "_id": "my_rep",
    "source":  "http://myserver.com:5984/foo",
    "target":  "bar",
    "create_target":  true,
    "_replication_id":  "c0ebe9256695ff083347cbf95f93e280",
    "_replication_state":  "triggered",
    "_replication_state_time":  1297974122
}
```

Special fields set by the replicator start with the prefix `_replication_`.

- `_replication_id`

  The ID internally assigned to the replication. This is also the ID exposed by `/_active_tasks`.

- `_replication_state`

  The current state of the replication.

- `_replication_state_time`

  A Unix timestamp (number of seconds since 1 Jan 1970) that tells us when the current replication state (marked in `_replication_state`) was set.

When the replication finishes, it will update the `_replication_state` field (and `_replication_state_time`) with the value `completed`, so the document will look like:

```
{
    "_id": "my_rep",
    "source":  "http://myserver.com:5984/foo",
    "target":  "bar",
    "create_target":  true,
    "_replication_id":  "c0ebe9256695ff083347cbf95f93e280",
    "_replication_state":  "completed",
    "_replication_state_time":  1297974122
}
```

When an error happens during replication, the `_replication_state` field is set to `error` (and `_replication_state_time` gets updated of course).

When you PUT/POST a document to the `_replicator` database, CouchDB will attempt to start the replication up to 10 times (configurable under `[replicator]`, parameter `max_replication_retry_count`). If it fails on the first attempt, it waits 5 seconds before doing a second attempt. If the second attempt fails, it waits 10 seconds before doing a third attempt. If the third attempt fails, it waits 20 seconds before doing a fourth attempt (each attempt doubles the previous wait period). When an attempt fails, the Couch log will show you something like:

```
[error] [<0.149.0>] Error starting replication `67c1bb92010e7abe35d7d629635f18b6+create_target` (
```

Note: The `_replication_state` field is only set to `error` when all the attempts were unsuccessful.

There are only 3 possible values for the `_replication_state` field: `triggered`, `completed` and `error`. Continuous replications never get their state set to `completed`.

5.2 Documents describing the same replication

Lets suppose 2 documents are added to the `_replicator` database in the following order:

```
{
    "_id": "doc_A",
    "source":  "http://myserver.com:5984/foo",
    "target":  "bar"
}
```

and

```
{
    "_id": "doc_B",
    "source":  "http://myserver.com:5984/foo",
    "target":  "bar"
}
```

Both describe exactly the same replication (only their `_ids` differ). In this case document `doc_A` triggers the replication, getting updated by CouchDB with the fields `_replication_state`, `_replication_state_time` and `_replication_id`, just like it was described before. Document `doc_B` however, is only updated with one field, the `_replication_id` so it will look like this:

```
{
    "_id": "doc_B",
    "source":  "http://myserver.com:5984/foo",
```

```
    "target":  "bar",
    "_replication_id":  "c0ebe9256695ff083347cbf95f93e280"
}
```

While document `doc_A` will look like this:

```
{
    "_id": "doc_A",
    "source":  "http://myserver.com:5984/foo",
    "target":  "bar",
    "_replication_id":  "c0ebe9256695ff083347cbf95f93e280",
    "_replication_state":  "triggered",
    "_replication_state_time":  1297974122
}
```

Note that both document get exactly the same value for the `_replication_id` field. This way you can identify which documents refer to the same replication - you can for example define a view which maps replication IDs to document IDs.

5.3 Canceling replications

To cancel a replication simply `DELETE` the document which triggered the replication. The Couch log will show you an entry like the following:

```
[Thu, 17 Feb 2011 20:16:29 GMT] [info] [<0.125.0>] Stopped replication `c0ebe9256695ff083347cbf95
```

Note: You need to `DELETE` the document that triggered the replication. `DELETE`-ing another document that describes the same replication but did not trigger it, will not cancel the replication.

5.4 Server restart

When CouchDB is restarted, it checks its `_replicator` database and restarts any replication that is described by a document that either has its `_replication_state` field set to `triggered` or it doesn't have yet the `_replication_state` field set.

Note: Continuous replications always have a `_replication_state` field with the value `triggered`, therefore they're always restarted when CouchDB is restarted.

5.5 Changing the Replicator Database

Imagine your replicator database (default name is `_replicator`) has the two following documents that represent pull replications from servers A and B:

```
{
    "_id": "rep_from_A",
    "source":  "http://aserver.com:5984/foo",
    "target":  "foo_a",
    "continuous":  true,
    "_replication_id":  "c0ebe9256695ff083347cbf95f93e280",
    "_replication_state":  "triggered",
    "_replication_state_time":  1297971311
}
```

```
{
    "_id": "rep_from_B",
    "source":  "http://bserver.com:5984/foo",
    "target":  "foo_b",
    "continuous":  true,
    "_replication_id":  "231bb3cf9d48314eaa8d48a9170570d1",
    "_replication_state":  "triggered",
    "_replication_state_time":  1297974122
}
```

Now without stopping and restarting CouchDB, you change the name of the replicator database to `another_replicator_db`:

```
$ curl -X PUT http://localhost:5984/_config/replicator/db -d '"another_replicator_db"'
"_replicator"
```

As soon as this is done, both pull replications defined before, are stopped. This is explicitly mentioned in CouchDB's log:

```
[Fri, 11 Mar 2011 07:44:20 GMT] [info] [<0.104.0>] Stopping all ongoing replications because the
[Fri, 11 Mar 2011 07:44:20 GMT] [info] [<0.127.0>] 127.0.0.1 - - PUT /_config/replicator/db 200
```

Imagine now you add a replication document to the new replicator database named `another_replicator_db`:

```
{
    "_id": "rep_from_X",
    "source":  "http://xserver.com:5984/foo",
    "target":  "foo_x",
    "continuous":  true
}
```

From now own you have a single replication going on in your system: a pull replication pulling from server X. Now you change back the replicator database to the original one `_replicator`:

```
$ curl -X PUT http://localhost:5984/_config/replicator/db -d '"_replicator"'
"another_replicator_db"
```

Immediately after this operation, the replication pulling from server X will be stopped and the replications defined in the `_replicator` database (pulling from servers A and B) will be resumed.

Changing again the replicator database to `another_replicator_db` will stop the pull replications pulling from servers A and B, and resume the pull replication pulling from server X.

5.6 Replicating the replicator database

Imagine you have in server C a replicator database with the two following pull replication documents in it:

```
{
     "_id": "rep_from_A",
     "source":  "http://aserver.com:5984/foo",
     "target":  "foo_a",
     "continuous":  true,
     "_replication_id":  "c0ebe9256695ff083347cbf95f93e280",
     "_replication_state":  "triggered",
     "_replication_state_time":  1297971311
}

{
     "_id": "rep_from_B",
     "source":  "http://bserver.com:5984/foo",
     "target":  "foo_b",
```

```
    "continuous":  true,
    "_replication_id":  "231bb3cf9d48314eaa8d48a9170570d1",
    "_replication_state":  "triggered",
    "_replication_state_time":  1297974122
}
```

Now you would like to have the same pull replications going on in server D, that is, you would like to have server D pull replicating from servers A and B. You have two options:

- Explicitly add two documents to server's D replicator database
- Replicate server's C replicator database into server's D replicator database

Both alternatives accomplish exactly the same goal.

5.7 Delegations

Replication documents can have a custom `user_ctx` property. This property defines the user context under which a replication runs. For the old way of triggering replications (POSTing to `/_replicate/`), this property was not needed (it didn't exist in fact) - this is because at the moment of triggering the replication it has information about the authenticated user. With the replicator database, since it's a regular database, the information about the authenticated user is only present at the moment the replication document is written to the database - the replicator database implementation is like a `_changes` feed consumer (with `?include_docs=true`) that reacts to what was written to the replicator database - in fact this feature could be implemented with an external script/program. This implementation detail implies that for non admin users, a `user_ctx` property, containing the user's name and a subset of his/her roles, must be defined in the replication document. This is ensured by the document update validation function present in the default design document of the replicator database. This validation function also ensure that a non admin user can set a user name property in the `user_ctx` property that doesn't match his/her own name (same principle applies for the roles).

For admins, the `user_ctx` property is optional, and if it's missing it defaults to a user context with name null and an empty list of roles - this mean design documents will not be written to local targets. If writing design documents to local targets is desired, the a user context with the roles `_admin` must be set explicitly.

Also, for admins the `user_ctx` property can be used to trigger a replication on behalf of another user. This is the user context that will be passed to local target database document validation functions.

Note: The `user_ctx` property only has effect for local endpoints.

Example delegated replication document:

```
{
    "_id": "my_rep",
    "source":  "http://bserver.com:5984/foo",
    "target":  "bar",
    "continuous":  true,
    "user_ctx": {
        "name": "joe",
        "roles": ["erlanger", "researcher"]
    }
}
```

As stated before, for admins the `user_ctx` property is optional, while for regular (non admin) users it's mandatory. When the roles property of `user_ctx` is missing, it defaults to the empty list `[ ]`.

CHAPTER

SIX

DESIGN DOCS

In this section we'll show how to write design documents, using the built-in *JavaScript Query Server*.

But before we start to write our first function, let's take a look at the list of common objects that will be used during our code journey - we'll be using them extensively within each function:

- *Database information object*
- *Request object*
- *Response object*
- *UserCtx object*
- *Database Security object*
- *Guide to JavaScript Query Server*

6.1 View functions

Views are the primary tool used for querying and reporting on CouchDB databases.

6.1.1 Map functions

mapfun(*doc*)

Arguments

- **doc** – Processed document object.

Map functions accept a single document as the argument and (optionally) `emit()` key/value pairs that are stored in a view.

```
function (doc) {
  if (doc.type === 'post' && doc.tags && Array.isArray(doc.tags)) {
    doc.tags.forEach(function (tag) {
      emit(tag.toLowerCase(), 1);
    });
  }
}
```

In this example a key/value pair is emitted for each value in the *tags* array of a document with a *type* of "post". Note that `emit()` may be called many times for a single document, so the same document may be available by several different keys.

Also keep in mind that each document is *sealed* to prevent situation when one map function changes document state and the other one received a modified version.

For efficiency reasons, documents are passed to a group of map functions - each document is processed by group of map functions from all views of related design document. This means that if you trigger index update for one view in ddoc, all others will get updated too.

Since *1.1.0* release *map* function supports *CommonJS* modules and access to `require()` function.

6.1.2 Reduce and rereduce functions

redfun(*keys*, *values*[, *rereduce*])

Arguments

- **keys** – Array of pairs docid-key for related map function result. Always `null` if rereduce is running (has `true` value).
- **values** – Array of map function result values.
- **rereduce** – Boolean sign of rereduce run.

Returns Reduces *values*

Reduce functions takes two required arguments of keys and values lists - the result of the related map function - and optional third one which indicates if *rereduce* mode is active or not. *Rereduce* is using for additional reduce values list, so when it is `true` there is no information about related *keys* (first argument is `null`).

Note, that if produced result by *reduce* function is longer than initial values list then a Query Server error will be raised. However, this behavior could be disabled by setting `reduce_limit` config option to `false`:

```
[query_server_config]
reduce_limit = false
```

While disabling `reduce_limit` might be useful for debug proposes, remember, that main task of reduce functions is to *reduce* mapped result, not to make it even bigger. Generally, your reduce function should converge rapidly to a single value - which could be an array or similar object.

Also CouchDB has three built-in reduce functions. These are implemented in Erlang and run right inside CouchDB, so they are much faster than the equivalent JavaScript functions: `_sum`, `_count` and `_stats`. Their equivalents in JavaScript below:

```
// could be replaced by _sum
function(keys, values){
  sum(values);
}

// could be replaced by _count
function(keys, values, rereduce){
  if (rereduce) {
    return sum(values);
  } else {
    return values.length;
  }
}

// could be replaced by _stats
function(keys, values, rereduce){
  return {
    'sum': sum(values),
    'min': Math.min.apply(null, values),
    'max': Math.max.apply(null, values),
    'count': values.length,
    'sumsqr': (function(){
      var sumsqr = 0;

      values.forEach(function (value) {
```

```
        sumsqr += value * value;
      });

      return sumsqr;
    })(),
  }
}
```

Note: Why don't reduce functions support CommonJS modules?

While *map* functions have limited access to stored modules through `require()` function there is no such feature for *reduce* functions. The reason lies deep inside in mechanism how *map* and *reduce* functions are processed by Query Server. Let's take a look on *map* functions first:

1. CouchDB sends all *map* functions for processed design document to Query Server.
2. Query Server handles them one by one, compiles and puts them onto an internal stack.
3. After all *map* functions had been processed, CouchDB will send the remaining documents to index one by one.
4. The Query Server receives the document object and applies it to every function from the stack. The emitted results are then joined into a single array and sent back to CouchDB.

Now let's see how *reduce* functions are handled:

1. CouchDB sends *as single command* list of available *reduce* functions with result list of key-value pairs that was previously received as result of *map* functions work.
2. Query Server compiles reduce functions and applies them to key-value lists. Reduced result sends back to CouchDB.

As you may note, *reduce* functions been applied in single shot while *map* ones are applied in an iterative way per each document. This means that it's possible for *map* functions to precompile CommonJS libraries and use them during the entire view processing, but for *reduce* functions it will be compiled again and again for each view result reduction, which will lead to performance degradation (*reduce* function are already does hard work to make large result smaller).

6.2 Show functions

showfun(*doc*, *req*)

Arguments

- **doc** – Processed document, may be omitted.
- **req** – *Request object*.

Returns *Response object*

Return type object or string

Show functions are used to represent documents in various formats, commonly as HTML page with nicer formatting. They can also be used to run server-side functions without requiring a pre-existing document.

Basic example of show function could be:

```
function(doc, req){
  if (doc) {
    return "Hello from " + doc._id + "!";
  } else {
    return "Hello, world!";
  }
}
```

Also, there is more simple way to return json encoded data:

```
function(doc, req){
  return {
    'json': {
      'id': doc['_id'],
      'rev': doc['_rev']
    }
  }
}
```

and even files (this one is CouchDB logo):

```
function(doc, req){
  return {
    'headers': {
      'Content-Type' : 'image/png',
    },
    'base64': ''.concat(
      'iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAMAAAAoLQ9TAAAAsV',
      'BMVEUAAAD//////////////////////5ur3rEBn/////////////////wDBL/',
      'AADuBAe9EB3IEBz/7+//X1/qBQn2AgP/f3/ilpzsDxfpChDtDhXeCA76AQH/v7',
      '/84eLyWV/uc3bJPEf/Dw/uw8bRWmP1h4zxSlD6YGHuQ0f6g4XyQkXvCA36MDH6',
      'wMH/z8/yAwX64ODeh47BHiv/Ly/20dLQLTj98PDXWmP/Pz//39/wGyJ7Iy9JAA',
      'AADHRSTlMAbw8vf08/bz+Pv19jK/W3AAAAg0lEQVR4Xp3LRQ4DQRBD0QqTm4Y5',
      'zMxw/4OleiJlHeUtv2X6RbNO1Uqj9g0RMCuQO0vBIg4vMFeOpCWIWmDOw82fZx',
      'vaND1c8OG4vrdOqD8YwgpDYDxRgkSm5rwu0nQVBJuMg++pLXZyr5jnc1BaH4GT',
      'LvEliY253nA3pVhQqdPt0f/erJkMGMB8xucAAAAASUVORK5CYII=')
  }
}
```

But what if you need to represent data in different formats via a single function? Functions `registerType()` and `provides()` are your the best friends in that question:

```
function(doc, req){
  provides('json', function(){
    return {'json': doc}
  });
  provides('html', function(){
    return '<pre>' + toJSON(doc) + '</pre>'
  })
  provides('xml', function(){
    return {
      'headers': {'Content-Type': 'application/xml'},
      'body' : ''.concat(
        '<?xml version="1.0" encoding="utf-8"?>\n',
        '<doc>',
        (function(){
          escape = function(s){
            return s.replace(/&quot;/g, '"')
                    .replace(/&gt;/g, '>')
                    .replace(/&lt;/g, '<')
                    .replace(/&amp;/g, '&');
          };
          var content = '';
          for(var key in doc){
            if(!doc.hasOwnProperty(key)) continue;
            var value = escape(toJSON(doc[key]));
            var key = escape(key);
            content += ''.concat(
              '<' + key + '>',
              value
              '</' + key + '>'
            )
```

```
          }
          return content;
        })(),
        '</doc>'
      )
    }
  })
  registerType('text-json', 'text/json')
  provides('text-json', function(){
    return toJSON(doc);
  })
}
```

This function may return *html*, *json* , *xml* or our custom *text json* format representation of same document object with same processing rules. Probably, the *xml* provider in our function needs more care to handle nested objects correctly, and keys with invalid characters, but you've got the idea!

See Also:

CouchDB Wiki:

- Showing Documents

CouchDB Guide:

- Show Functions

6.3 List functions

listfun (*head*, *req*)

Arguments

- **head** – *View Head Information*
- **req** – *Request object*.

Returns Last chunk.

Return type string

While *Show functions* are used to customize document presentation, *List functions* are used for same purpose, but against *View functions* results.

The next list function formats view and represents it as a very simple HTML page:

```
function(head, req){
  start({
    'headers': {
      'Content-Type': 'text/html'
    }
  });
  send('<html><body><table>');
  send('<tr><th>ID</th><th>Key</th><th>Value</th></tr>')
  while(row = getRow()){
    send(''.concat(
      '<tr>',
      '<td>' + toJSON(row.id) + '</td>',
      '<td>' + toJSON(row.key) + '</td>',
      '<td>' + toJSON(row.value) + '</td>',
      '</tr>'
    ));
  }
  send('</table></body></html>');
}
```

Templates and styles could obviously be used to present data in a nicer fashion, but this is an excellent starting point. Note that you may also use `registerType()` and `provides()` functions in the same way as for *Show functions*!

See Also:

CouchDB Wiki:

- Listing Views with CouchDB 0.10 and later

CouchDB Guide:

- Transforming Views with List Functions

6.4 Update functions

updatefun(*doc*, *req*)

Arguments

- **doc** – Update function target document.
- **req** – *Request object*

Returns Two-element array: the first element is the (updated or new) document, which is committed to the database. If the first element is `null` no document will be committed to the database. If you are updating an existing, it should already have an `_id` set, and if you are creating a new document, make sure to set its `_id` to something, either generated based on the input or the `req.uuid` provided. The second element is the response that will be sent back to the caller.

Update handlers are functions that clients can request to invoke server-side logic that will create or update a document. This feature allows a range of use cases such as providing a server-side last modified timestamp, updating individual fields in a document without first getting the latest revision, etc.

When the request to an update handler includes a document ID in the URL, the server will provide the function with the most recent version of that document. You can provide any other values needed by the update handler function via the `POST`/`PUT` entity body or query string parameters of the request.

The basic example that demonstrates all use-cases of update handlers below:

```
function(doc, req){
    if (!doc){
        if ('id' in req){
            // create new document
            return [{'_id': req['id']}, 'New World']
        }
        // change nothing in database
        return [null, 'Empty World']
    }
    doc['world'] = 'hello';
    doc['edited_by'] = req['userCtx']['name']
    return [doc, 'Edited World!']
}
```

See Also:

CouchDB Wiki:

- Document Update Handlers

6.5 Filter functions

filterfun(*doc*, *req*)

Arguments

- **doc** – Processed document object.
- **req** – *Request object*

Returns Boolean value: `true` means that *doc* passes the filter rules, `false` that not.

Filter functions are mostly acts like *Show functions* and *List functions*: they formats, but more correctly to say, they *filters changes feed*.

6.5.1 Classic filters

By default the changes feed emits all database documents changes. But if you're waiting for some special changes, processing all documents is inefficient.

Filters are special design document functions that allows changes feed to emit only specific documents that pass filter rules.

Lets assume that our database is a mailbox and we need to to handle only new mails (documents with status *new*) events. Assuming that, our filter function will looks like next one:

```
function(doc, req){
  // we need only `mail` documents
  if (doc.type != 'mail'){
    return false;
  }
  // we're interested only in `new` ones
  if (doc.status != 'new'){
    return false;
  }
  return true; // passed!
}
```

Filter functions must return `true` in fact if document passed all defined rules. Now, if you apply this function to changes feed it will emit only changes about "new mails":

```
GET /somedatabase/_changes?filter=mailbox/new_mail HTTP/1.1
```

```
{"results":[
{"seq":1,"id":"df8eca9da37dade42ee4d7aa3401f1dd","changes":[{"rev":"1-c2e0085a21d34fa1cecb6dc26a4
{"seq":7,"id":"df8eca9da37dade42ee4d7aa34024714","changes":[{"rev":"1-29d748a6e87b43db967fe338bcb
],
"last_seq":27}
```

Note, that `last_seq` number is 27, but we'd received only two records. Seems like any other changes was about documents that hasn't passed our filter.

Probably, we also need to filter changes feed of our mailbox not only by single status value: we're also interested in statuses like "spam" to update spam-filter heuristic rules, "outgoing" to let mail daemon actually send mails and so on. Creating a lot of similar functions that actually does similar work isn't good idea - so we need dynamic filter to go.

If you have noted, filter functions takes second argument as *request* object - it allows to create dynamic filters based on query parameters, *user context* and more.

The dynamic version of our filter now will be next:

```
function(doc, req){
  // we need only `mail` documents
  if (doc.type != 'mail'){
    return false;
  }
  // we're interested only in requested status
  if (doc.status != req.query.status){
```

```
    return false;
  }
  return true; // passed!
}
```

and now we have pass *status* query parameter in request to let filter match only required documents:

```
GET /somedatabase/_changes?filter=mailbox/by_status&status=new HTTP/1.1
```


```
{"results":[
{"seq":1,"id":"df8eca9da37dade42ee4d7aa3401f1dd","changes":[{"rev":"1-c2e0085a21d34fa1cecb6dc26a4
{"seq":7,"id":"df8eca9da37dade42ee4d7aa34024714","changes":[{"rev":"1-29d748a6e87b43db967fe338bcb
],
"last_seq":27}
```


and we can change filter behavior with easy:

```
GET /somedatabase/_changes?filter=mailbox/by_status&status=spam HTTP/1.1
```


```
{"results":[
{"seq":11,"id":"8960e91220798fc9f9d29d24ed612e0d","changes":[{"rev":"3-cc6ff71af716ddc2ba11496702
],
"last_seq":27}
```


Combining filters with *continuous* feed allows to create powerful event-driven systems.

6.5.2 View filters

View filters are the same as above, with one small difference: they use views *map* function instead to *filter* one to process the changes feed. Each time when a key-value pair could be emitted, a change is returned. This allows to avoid creating filter functions that are mostly does same works as views.

To use them just specify *_view* value for `filter` parameter and *designdoc/viewname* for `view` one:

```
GET /somedatabase/_changes?filter=_view&view=dname/viewname  HTTP/1.1
```

Note: Since view filters uses *map* functions as filters, they can't show any dynamic behavior since *request object* is not available.

See Also:

CouchDB Guide:

- Guide to filter change notification

CouchDB Wiki:

- Filtered replication

6.6 Validate document update functions

validatefun(*newDoc*, *oldDoc*, *userCtx*, *secObj*)

Arguments

- **newDoc** – New version of document that will be stored.
- **oldDoc** – Previous version of document that is already stored.
- **userCtx** – *User Context Object*
- **secObj** – *Security Object*

Throws `forbidden` error to gracefully prevent document storing.

Throws `unauthorized` error to prevent storage and allow the user to re-auth.

A design document may contain a function named *validate_doc_update* which can be used to prevent invalid or unauthorized document update requests from being stored. The function is passed the new document from the update request, the current document stored in the database, a *User Context Object* containing information about the user writing the document (if present), and a *Security Object* with lists of database security roles.

Validation functions typically examine the structure of the new document to ensure that required fields are present and to verify that the requesting user should be allowed to make changes to the document properties. For example, an application may require that a user must be authenticated in order to create a new document or that specific document fields be present when a document is updated. The validation function can abort the pending document write by throwing one of two error objects:

```
// user is not authorized to make the change but may re-authenticate
throw({ unauthorized: 'Error message here.' });

// change is not allowed
throw({ forbidden: 'Error message here.' });
```

Document validation is optional, and each design document in the database may have at most one validation function. When a write request is received for a given database, the validation function in each design document in that database is called in an unspecified order. If any of the validation functions throw an error, the write will not succeed.

Example: The `_design/_auth` ddoc from *_users* database uses a validation function to ensure that documents contain some required fields and are only modified by a user with the `_admin` role:

```
function(newDoc, oldDoc, userCtx, secObj) {
    if (newDoc._deleted === true) {
        // allow deletes by admins and matching users
        // without checking the other fields
        if ((userCtx.roles.indexOf('_admin') !== -1) ||
            (userCtx.name == oldDoc.name)) {
            return;
        } else {
            throw({forbidden: 'Only admins may delete other user docs.'});
        }
    }

    if ((oldDoc && oldDoc.type !== 'user') || newDoc.type !== 'user') {
        throw({forbidden : 'doc.type must be user'});
    } // we only allow user docs for now

    if (!newDoc.name) {
        throw({forbidden: 'doc.name is required'});
    }

    if (!newDoc.roles) {
        throw({forbidden: 'doc.roles must exist'});
    }

    if (!isArray(newDoc.roles)) {
        throw({forbidden: 'doc.roles must be an array'});
    }

    if (newDoc._id !== ('org.couchdb.user:' + newDoc.name)) {
        throw({
            forbidden: 'Doc ID must be of the form org.couchdb.user:name'
        });
    }

    if (oldDoc) { // validate all updates
```

```
        if (oldDoc.name !== newDoc.name) {
            throw({forbidden: 'Usernames can not be changed.'});
        }
    }

    if (newDoc.password_sha && !newDoc.salt) {
        throw({
            forbidden: 'Users with password_sha must have a salt.' +
                'See /_utils/script/couch.js for example code.'
        });
    }

    var is_server_or_database_admin = function(userCtx, secObj) {
        // see if the user is a server admin
        if(userCtx.roles.indexOf('_admin') !== -1) {
            return true; // a server admin
        }

        // see if the user a database admin specified by name
        if(secObj && secObj.admins && secObj.admins.names) {
            if(secObj.admins.names.indexOf(userCtx.name) !== -1) {
                return true; // database admin
            }
        }

        // see if the user a database admin specified by role
        if(secObj && secObj.admins && secObj.admins.roles) {
            var db_roles = secObj.admins.roles;
            for(var idx = 0; idx < userCtx.roles.length; idx++) {
                var user_role = userCtx.roles[idx];
                if(db_roles.indexOf(user_role) !== -1) {
                    return true; // role matches!
                }
            }
        }

        return false; // default to no admin
    }

    if (!is_server_or_database_admin(userCtx, secObj)) {
        if (oldDoc) { // validate non-admin updates
            if (userCtx.name !== newDoc.name) {
                throw({
                    forbidden: 'You may only update your own user document.'
                });
            }
            // validate role updates
            var oldRoles = oldDoc.roles.sort();
            var newRoles = newDoc.roles.sort();

            if (oldRoles.length !== newRoles.length) {
                throw({forbidden: 'Only _admin may edit roles'});
            }

            for (var i = 0; i < oldRoles.length; i++) {
                if (oldRoles[i] !== newRoles[i]) {
                    throw({forbidden: 'Only _admin may edit roles'});
                }
            }
        } else if (newDoc.roles.length > 0) {
            throw({forbidden: 'Only _admin may set roles'});
        }
    }
```

```
    // no system roles in users db
    for (var i = 0; i < newDoc.roles.length; i++) {
        if (newDoc.roles[i][0] === '_') {
            throw({
                forbidden:
                'No system roles (starting with underscore) in users db.'
            });
        }
    }

    // no system names as names
    if (newDoc.name[0] === '_') {
        throw({forbidden: 'Username may not start with underscore.'});
    }

    var badUserNameChars = [':'];

    for (var i = 0; i < badUserNameChars.length; i++) {
        if (newDoc.name.indexOf(badUserNameChars[i]) >= 0) {
            throw({forbidden: 'Character `' + badUserNameChars[i] +
                    '` is not allowed in usernames.'});
        }
    }
}
```

Note: The `return` statement used only for function, it has no impact on the validation process.

See Also:

CouchDB Guide:

- Validation Functions

CouchDB Wiki:

- Document Update Validation

CHAPTER

SEVEN

QUERY SERVERS

7.1 JavaScript

Note: While every design function has access to all JavaScript objects, the table below describes appropriate usage cases. For example, you may use `emit()` in *List functions*, but `getRow()` is not permitted during *Map functions*.

JS Function	Reasonable to use in design doc functions
`emit()`	*Map functions*
`getRow()`	*List functions*
`JSON`	any
`isArray()`	any
`log()`	any
`provides()`	*Show functions*, *List functions*
`registerType()`	*Show functions*, *List functions*
`require()`	any, except *Reduce and rereduce functions*
`send()`	*List functions*
`start()`	*List functions*
`sum()`	any
`toJSON()`	any

7.1.1 Design functions context

Each design function executes in a special context of predefined objects, modules and functions:

emit(*key*, *value*)

Emits a *key-value* pair for further processing by CouchDB after the map function is done.

Arguments

- **key** – The view key
- **value** – The *key*'s associated value

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

getRow()

Extracts the next row from a related view result.

Returns View result row

Return type object

```
function(head, req){
  send('[');
  row = getRow();
  if (row){
    send(toJSON(row));
    while(row = getRow()){
      send(',');
      send(toJSON(row));
    }
  }
  return ']';
}
```

JSON

JSON2 object.

isArray(*obj*)

A helper function to check if the provided value is an *Array*.

Arguments

- **obj** – Any Javascript value

Returns `true` if *obj* is *Array*-typed, `false` otherwise

Return type boolean

log(*message*)

Log a message to the CouchDB log (at the *INFO* level).

Arguments

- **message** – Message to be logged

```
function(doc){
  log('Procesing doc ' + doc['_id']);
  emit(doc['_id'], null);
}
```

After the map function has run, the following line can be found in CouchDB logs (e.g. at */var/log/couchdb/couch.log*):

```
[Sat, 03 Nov 2012 17:38:02 GMT] [info] [<0.7543.0>] OS Process #Port<0.3289> Log :: Processin
```

provides(*key*, *func*)

Registers callable handler for specified MIME key.

Arguments

- **key** – MIME key previously defined by `registerType()`
- **func** – MIME type handler

registerType(*key*, **mimes*)

Registers list of MIME types by associated *key*.

Arguments

- **key** – MIME types
- **mimes** – MIME types enumeration

Predefined mappings (*key-array*):

- **all**: `*/*`
- **text**: `text/plain; charset=utf-8, txt`
- **html**: `text/html; charset=utf-8`

- **xhtml**: `application/xhtml+xml`, `xhtml`
- **xml**: `application/xml`, `text/xml`, `application/x-xml`
- **js**: `text/javascript`, `application/javascript`, `application/x-javascript`
- **css**: `text/css`
- **ics**: `text/calendar`
- **csv**: `text/csv`
- **rss**: `application/rss+xml`
- **atom**: `application/atom+xml`
- **yaml**: `application/x-yaml`, `text/yaml`
- **multipart_form**: `multipart/form-data`
- **url_encoded_form**: `application/x-www-form-urlencoded`
- **json**: `application/json`, `text/x-json`

require(*path*)

Loads CommonJS module by a specified *path*. The path should not start with a slash.

Arguments

- **path** – A CommonJS module path started from design document root

Returns Exported statements

send(*chunk*)

Sends a single string *chunk* in response.

Arguments

- **chunk** – Text chunk

```
function(head, req){
  send('Hello,');
  send(' ');
  send('Couch');
  return !
}
```

start(*init_resp*)

Initiates chunked response. As an option, a custom *response* object may be sent at this point. For *list*-functions only!

Note: list functions may set the *HTTP response code* and *headers* by calling this function. This function must be called before `send()`, `getRow()` or a *return* statement; otherwise, the query server will implicitly call this function with the empty object (`{}`).

```
function(head, req){
  start({
    "code": 302,
    "headers": {
      "Location": "http://couchdb.apache.org"
    }
  });
  return "Relax!";
}
```

sum(*arr*)

Sum *arr*'s items.

Arguments

- **arr** – Array of numbers

Return type number

toJSON(*obj*)

Encodes *obj* to JSON string. This is an alias for the `JSON.stringify` method.

Arguments

- **obj** – JSON encodable object

Returns JSON string

7.1.2 CommonJS Modules

Support for CommonJS Modules (introduced in CouchDB 0.11.0) allows you to create modular design functions without the need for duplication of functionality.

Here's a CommonJS module that checks user permissions:

```
function user_context(userctx, secobj) {
  var is_admin = function() {
    return userctx.indexOf('_admin') != -1;
  }
  return {'is_admin': is_admin}
}

exports['user'] = user_context
```

Each module has access to additional global variables:

- **module** (*object*): Contains information about the stored module
  - **id** (*string*): The module id; a JSON path in ddoc context
  - **current** (*code*): Compiled module code object
  - **parent** (*object*): Parent frame
  - **exports** (*object*): Export statements
- **exports** (*object*): Shortcut to the `module.exports` object

The CommonJS module can be added to a design document, like so:

```
{
   "views": {
      "lib": {
         "security": "function user_context(userctx, secobj) { ... }"
      },
      "validate_doc_update": "function(newdoc, olddoc, userctx, secobj) {
        user = require('lib/security').user(userctx, secobj);
        return user.is_admin();
      }"
   },
   "_id": "_design/test"
}
```

Modules paths are relative to the design document's `views` object, but modules can only be loaded from the object referenced via `lib`. The `lib` structure can still be used for view functions as well, by simply storing view functions at e.g. `views.lib.map`, `views.lib.reduce`, etc.

7.2 Erlang

Warning: Unlike the JavaScript query server, the Erlang query server does not run in a sandbox. This means Erlang code has full access to your OS, filesystem and network, which may lead to security issues. While Erlang functions are faster than JavaScript ones, you need to be careful about running them, especially if they were written by someone else.

Note: Due to security restrictions, the Erlang query server is disabled by default. To enable it, you need to edit your *local.ini* to include a `native_query_servers` section:

```
[native_query_servers]
erlang = {couch_native_process, start_link, []}
```

Don't forget to restart CouchDB after updating the configuration, and use the `language: "erlang"` property in your Erlang design documents.

Emit (*Id*, *Value*)

Emits *key-value* pairs to view indexer process.

```
fun({Doc}) ->
  <<K,_/binary>> = proplists:get_value(<<"_rev">>, Doc, null),
  V = proplists:get_value(<<"_id">>, Doc, null),
  Emit(<<K>>, V)
end.
```

FoldRows (*Fun*, *Acc*)

Helper to iterate over all rows in a list function.

Arguments

- **Fun** – Function object.
- **Acc** – The value previously returned by *Fun*.

```
fun(Head, {Req}) ->
  Fun = fun({Row}, Acc) ->
    Id = couch_util:get_value(<<"id">>, Row),
    Send(list_to_binary(io_lib:format("Previous doc id: ~p~n", [Acc]))),
    Send(list_to_binary(io_lib:format("Current  doc id: ~p~n", [Id]))),
    {ok, Id}
  end,
  FoldRows(Fun, nil),
  ""
end.
```

GetRow ()

Retrieves the next row from a related view result.

```
%% FoldRows background implementation.
%% https://git-wip-us.apache.org/repos/asf?p=couchdb.git;a=blob;f=src/couchdb/couch_native_pr
%%
foldrows(GetRow, ProcRow, Acc) ->
  case GetRow() of
    nil ->
      {ok, Acc};
    Row ->
      case (catch ProcRow(Row, Acc)) of
        {ok, Acc2} ->
          foldrows(GetRow, ProcRow, Acc2);
        {stop, Acc2} ->
          {ok, Acc2}
```

```
        end
    end.
```

Log(*Msg*)

Arguments

- **Msg** – Log a message at the *INFO* level.

```
fun({Doc}) ->
  <<K,_/binary>> = proplists:get_value(<<"_rev">>, Doc, null),
  V = proplists:get_value(<<"_id">>, Doc, null),
  Log(lists:flatten(io_lib:format("Hello from ~s doc!", [V]))),
  Emit(<<K>>, V)
end.
```

After the map function has run, the following line can be found in CouchDB logs (e.g. at */var/log/couchdb/couch.log*):

```
[Sun, 04 Nov 2012 11:33:58 GMT] [info] [<0.9144.2>] Hello from 8d300b86622d67953d102165dbe994
```

Send(*Chunk*)

Sends a single string *Chunk* in response.

```
fun(Head, {Req}) ->
  Send("Hello,"),
  Send(" "),
  Send("Couch"),
  "!"
end.
```

The function above produces the following response:

```
Hello, Couch!
```

Start(*Headers*)

Arguments

- **Headers** – Proplist of *response object*.

Initialize *List functions* response. At this point, response code and headers may be defined. For example, this function redirects to the CouchDB web site:

```
fun(Head, {Req}) ->
  Start({[{<<"code">>, 302},
          {<<"headers">>, {[
            {<<"Location">>, <<"http://couchdb.apache.org">>}]
          }}
        ]}),
  "Relax!"
end.
```

CHAPTER

EIGHT

CHANGES FEED

8.1 Polling

A list of changes made to documents in the database, in the order they were made, can be obtained from the database's `_changes` resource. You can query the `_changes` resource by issuing a `GET` request with the following (optional) parameters:

Parameter	Value	Default Value	Notes
since	seqnum / now	0	(1)
limit	maxsequences	none	(2)
descending	boolean	false	(3)
feed	normal / longpoll / continuous / eventsource	normal	(4)
heartbeat	milliseconds	60000	(5)
timeout	milliseconds	60000	(6)
filter	designdoc/filtername / _view	none	(7)
include_docs	boolean	false	(8)
style	all_docs / main_only	main_only	(9)
view	designdoc/filtername	none	(10)

Notes:

1. Start the results from the change immediately after the given sequence number.
2. Limit number of result rows to the specified value (note that using 0 here has the same effect as 1).
3. Return the change results in descending sequence order (most recent change first)
4. Select the type of feed.
5. Period in milliseconds after which an empty line is sent in the results. Only applicable for *longpoll* or *continuous* feeds. Overrides any timeout to keep the feed alive indefinitely.
6. Maximum period in milliseconds to wait for a change before the response is sent, even if there are no results. Only applicable for *longpoll* or *continuous* feeds. Note that 60000 is also the default maximum timeout to prevent undetected dead connections.

   You can change the default maximum timeout in your ini-configuration:

   ```
   [httpd]
   changes_timeout=#millisecs
   ```

7. Reference to a *filter function* from a design document that will filter whole stream emitting only filtered events. See the section in the book for more information.
8. Include the associated document with each result. If there are conflicts, only the winning revision is returned.
9. Specifies how many revisions are returned in the changes array. The default, *main_only*, will only return the current "winning" revision; *all_docs* will return all leaf revisions (including conflicts and deleted former conflicts.)

10. Allows to use view functions as filters. It requires to set `filter` special value *_view* to enable this feature. Documents counted as "passed" for view filter in case if map function emits at least one record for them.

Changed in version 0.11.0: added `include_docs` parameterChanged in version 1.2.0: added `view` parameter and special value *_view* for `filter` oneChanged in version 1.3.0: `since` parameter could take *now* value to start listen changes since current seq number.Changed in version 1.3.0: `eventsource` feed type added. By default all changes are immediately returned as a JSON object:

```
GET /somedatabase/_changes HTTP/1.1
```

```
{"results":[
{"seq":1,"id":"fresh","changes":[{"rev":"1-967a00dff5e02add41819138abb3284d"}]},
{"seq":3,"id":"updated","changes":[{"rev":"2-7051cbe5c8faecd085a3fa619e6e6337"}]},
{"seq":5,"id":"deleted","changes":[{"rev":"2-eec205a9d413992850a6e32678485900"}],"deleted":true}
],
"last_seq":5}
```

`results` is the list of changes in sequential order. New and changed documents only differ in the value of the rev; deleted documents include the `"deleted": true` attribute. (In the `style=all_docs mode`, deleted applies only to the current/winning revision. The other revisions listed might be deleted even if there is no deleted property; you have to `GET` them individually to make sure.)

`last_seq` is the sequence number of the last update returned. (Currently it will always be the same as the seq of the last item in results.)

Sending a `since` param in the query string skips all changes up to and including the given sequence number:

```
GET /somedatabase/_changes?since=3 HTTP/1.1
```

```
{"results":[
{"seq":5,"id":"deleted","changes":[{"rev":"2-eec205a9d413992850a6e32678485900"}],"deleted":true}
],
"last_seq":5}
```

8.2 Long Polling

The *longpoll* feed (probably most useful used from a browser) is a more efficient form of polling that waits for a change to occur before the response is sent. *longpoll* avoids the need to frequently poll CouchDB to discover nothing has changed!

The response is basically the same JSON as is sent for the normal feed.

A timeout limits the maximum length of time the connection is open. If there are no changes before the timeout expires the response's results will be an empty list.

8.3 Continuous

Polling the CouchDB server is not a good thing to do. Setting up new HTTP connections just to tell the client that nothing happened puts unnecessary strain on CouchDB.

A continuous feed stays open and connected to the database until explicitly closed and changes are sent to the client as they happen, i.e. in near real-time.

The continuous feed's response is a little different than the other feed types to simplify the job of the client - each line of the response is either empty or a JSON object representing a single change, as found in the normal feed's results.

```
GET /somedatabase/_changes?feed=continuous HTTP/1.1
```

```
{"seq":1,"id":"fresh","changes":[{"rev":"1-967a00dff5e02add41819138abb3284d"}]}
{"seq":3,"id":"updated","changes":[{"rev":"2-7051cbe5c8faecd085a3fa619e6e6337"}]}
{"seq":5,"id":"deleted","changes":[{"rev":"2-eec205a9d413992850a6e32678485900"}],"deleted":true}
... tum tee tum ...
{"seq":6,"id":"updated","changes":[{"rev":"3-825cb35de44c433bfb2df415563a19de"}]}
```

Obviously, ... *tum tee tum* ... does not appear in the actual response, but represents a long pause before the change with seq 6 occurred.

8.4 Event Source

The *eventsource* feed provides push notifications that can be consumed in the form of DOM events in the browser. Refer to the W3C eventsource specification for further details. CouchDB honors the `Last-Event-ID` header, and if it's present it will take precedence over the `since` query parameter.

```
GET /somedatabase/_changes?feed=eventsource HTTP/1.1

// define the event handling function
if (window.EventSource) {

  var source = new EventSource("/somedatabase/_changes?feed=eventsource");
  source.onerror = function(e) {
    alert('EventSource failed.');
  };

  var results = [];
  var sourceListener = function(e) {
    var data = JSON.parse(e.data);
    results.push(data);
  };

  // start listening for events
  source.addEventListener('message', sourceListener, false);

  // stop listening for events
  source.removeEventListener('message', sourceListener, false);

}
```

Note: EventSource connections are subject to cross-origin resource sharing restrictions. You might need to use the experimental *CORS support* to get the EventSource to work in your application.

CHAPTER

NINE

API REFERENCE

The components of the API URL path help determine the part of the CouchDB server that is being accessed. The result is the structure of the URL request both identifies and effectively describes the area of the database you are accessing.

As with all URLs, the individual components are separated by a forward slash.

As a general rule, URL components and JSON fields starting with the _ (underscore) character represent a special component or entity within the server or returned object. For example, the URL fragment `/_all_dbs` gets a list of all of the databases in a CouchDB instance.

This reference is structured according to the URL structure, as below.

9.1 Database Methods

The Database methods provide an interface to an entire database withing CouchDB. These are database, rather than document, level requests.

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

Method	Path	Description
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/_compact	Starts a compaction for the database
POST	/db/_compact/design-doc	Starts a compaction for all the views in the selected design document
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/_purge	Purge some historical documents entirely from database history
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/_temp_view	Execute a given view function for all documents and return the result
POST	/db/_view_cleanup	Removes view files that are not used by any design document

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.

9.1.1 GET /db

- **Method**: `GET /db`
- **Request**: None
- **Response**: Information about the database in JSON format
- **Admin Privileges Required**: no
- **Return Codes**:
  - **404**: The requested content could not be found. The returned content will include further information, as a JSON object, if available.

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

```
GET http://couchdb:5984/recipes
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:

```
{
   "compact_running" : false,
   "committed_update_seq" : 375048,
   "disk_format_version" : 5,
   "disk_size" : 33153123,
   "doc_count" : 18386,
   "doc_del_count" : 0,
   "db_name" : "recipes",
   "instance_start_time" : "1290700340925570",
   "purge_seq" : 10,
   "update_seq" : 375048
}
```

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

Field	Description
committed_update_seq	The number of committed update.
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	Timestamp of when the database was opened, expressed in microseconds since the epoch.
purge_seq	The number of purge operations on the database.
update_seq	The current number of updates to the database.

9.1.2 PUT /db

- **Method**: PUT /db
- **Request**: None
- **Response**: JSON success statement
- **Admin Privileges Required**: no
- **Return Codes**:
  - **400**: 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 http://couchdb:5984/recipes
Content-Type: application/json
```

The returned content contains the JSON status:

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

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

9.1.3 DELETE /db

- **Method**: DELETE /db
- **Request**: None
- **Response**: JSON success statement
- **Admin Privileges Required**: no
- **Return Codes**:
  - **200**: Database has been deleted
  - **404**: The requested content could not be found. The returned content will include further information, as a JSON object, if available.

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

To delete the database recipes you would send the request:

```
DELETE http://couchdb:5984/recipes
Content-Type: application/json
```

If successful, the returned JSON will indicate success

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

9.1.4 GET /db/_changes

- **Method**: `GET /db/_changes`
- **Request**: None
- **Response**: JSON success statement
- **Admin Privileges Required**: no
- **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

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. 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

  ```
  DELETE http://couchdb:5984/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.

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. For more information, see *Changes Feed*.

9.1.5 POST /db/_compact

- **Method**: `POST /db/_compact`
- **Request**: None
- **Response**: JSON success statement
- **Admin Privileges Required**: yes
- **Return Codes**:
  - **202**: Compaction request has been accepted
  - **404**: The requested content could not be found. The returned content will include further information, as a JSON object, if available.

Request compaction of the specified database. Compaction compresses the disk database file by performing the following operations:

- Writes a new version of the database file, removing any unused sections from the new version during write. Because a new file is temporary created for this purpose, you will need twice the current storage space of the specified database in order for the compaction routine to complete.
- Removes old revisions of documents from the database, up to the per-database limit specified by the `_revs_limit` database parameter. See *GET /db*.

Compaction can only be requested on an individual database; you cannot compact all the databases for a CouchDB instance. The compaction process runs as a background process.

You can determine if the compaction process is operating on a database by obtaining the database meta information, the `compact_running` value of the returned database structure will be set to true. See *GET /db*.

You can also obtain a list of running processes to determine whether compaction is currently running. See *GET /_active_tasks*.

9.1.6 POST /db/_compact/design-doc

- **Method**: `POST /db/_compact/design-doc`
- **Request**: None
- **Response**: JSON success statement
- **Admin Privileges Required**: yes
- **Return Codes**:
  - **202**: Compaction request has been accepted

- **404**: The requested content could not be found. The returned content will include further information, as a JSON object, if available.

Compacts the view indexes associated with the specified design document. You can use this in place of the full database compaction if you know a specific set of view indexes have been affected by a recent database change.

For example, to compact the views associated with the `recipes` design document:

```
POST http://couchdb:5984/recipes/_compact/recipes
Content-Type: application/json
```

CouchDB will immediately return with a status indicating that the compaction request has been received (HTTP status code 202):

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

9.1.7 POST /db/_view_cleanup

- **Method**: `POST /db/_view_cleanup`
- **Request**: None
- **Response**: JSON success statement
- **Admin Privileges Required**: yes

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

```
POST http://couchdb:5984/recipes/_view_cleanup
Content-Type: application/json
```

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

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

9.1.8 POST /db/_ensure_full_commit

- **Method**: `POST /db/_ensure_full_commit`
- **Request**: None
- **Response**: JSON success statement
- **Admin Privileges Required**: no
- **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 http://couchdb:5984/recipes/_ensure_full_commit
Content-Type: application/json
```

This returns a status message, containing the success message and the timestamp for when the CouchDB instance was started:

```
{
  "ok" : true,
  "instance_start_time" : "1288186189373361"
}
```

9.1.9 POST /db/_bulk_docs

- **Method**: `POST /db/_bulk_docs`
- **Request**: JSON of the docs and updates to be applied
- **Response**: JSON success statement
- **Admin Privileges Required**: no
- **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
all_or_nothing (optional)	Sets the database commit mode to use all-or-nothing semantics
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

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, two with the supplied document IDs, and one which will have a document ID generated:

```
{
   "docs" : [
      {
         "_id" : "FishStew",
         "servings" : 4,
         "subtitle" : "Delicious with fresh bread",
         "title" : "Fish Stew"
      },
      {
         "_id" : "LambStew",
         "servings" : 6,
         "subtitle" : "Delicious with scone topping",
         "title" : "Lamb Stew"
      },
      {
         "servings" : 8,
         "subtitle" : "Delicious with suet dumplings",
         "title" : "Beef Stew"
      },
```

```
    ]
}
```

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:

```
POST http://couchdb:5984/recipes/_bulk_docs
Content-Type: application/json

[
    {
        "id" : "FishStew",
        "rev" : "1-9c65296036141e575d32ba9c034dd3ee",
    },
    {
        "id" : "LambStew",
        "rev" : "1-34c318924a8f327223eed702ddfdc66d",
    },
    {
        "id" : "7f7638c86173eb440b8890839ff35433",
        "rev" : "1-857c7cbeb6c8dd1dd34a0c73e8da3c44",
    }
]
```

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 http://couchdb:5984/recipes/_bulk_docs
Content-Type: application/json

{
    "docs" : [
        {
            "_id" : "FishStew",
            "_rev" : "1-9c65296036141e575d32ba9c034dd3ee",
            "servings" : 4,
            "subtitle" : "Delicious with freshly baked bread",
            "title" : "Fish Stew"
        },
        {
            "_id" : "LambStew",
            "_rev" : "1-34c318924a8f327223eed702ddfdc66d",
            "servings" : 6,
            "subtitle" : "Serve with a wholemeal scone topping",
            "title" : "Lamb Stew"
        },
        {
            "_id" : "7f7638c86173eb440b8890839ff35433"
            "_rev" : "1-857c7cbeb6c8dd1dd34a0c73e8da3c44",
            "servings" : 8,
            "subtitle" : "Hand-made dumplings make a great accompaniment",
            "title" : "Beef Stew"
```

```
        }
    ]
}
```

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

```
[
    {
        "id" : "FishStew",
        "rev" : "2-e7af4c4e9981d960ecf78605d79b06d1"
    },
    {
        "id" : "LambStew",
        "rev" : "2-0786321986194c92dd3b57dfbfc741ce"
    },
    {
        "id" : "7f7638c86173eb440b8890839ff35433",
        "rev" : "2-bdd3bf3563bee516b96885a66c743f8e"
    }
]
```

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

CouchDB supports two different modes for updating (or inserting) documents using the bulk documentation system. Each mode affects both the state of the documents in the event of system failure, and the level of conflict checking performed on each document. The two modes are:

- `non-atomic`

  The default mode is non-atomic, that is, CouchDB 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.

  In this mode, 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.

- `all-or-nothing`

  In all-or-nothing mode, either all documents are written to the database, or no documents are written to the database, in the event of a system failure during commit.

  In addition, the per-document conflict checking is not performed. Instead a new revision of the document is created, even if the new revision is in conflict with the current revision in the database. The returned structure contains the list of documents with new revisions:

```
[
    {
        "id" : "FishStew",
        "rev" : "2-e7af4c4e9981d960ecf78605d79b06d1"
    },
    {
        "id" : "LambStew",
        "rev" : "2-0786321986194c92dd3b57dfbfc741ce"
    },
    {
        "id" : "7f7638c86173eb440b8890839ff35433",
        "rev" : "2-bdd3bf3563bee516b96885a66c743f8e"
    }
]
```

  When updating documents using this mode the revision of a document included in views will be arbitrary. You can check the conflict status for a document by using the `conflicts=true` query argument when accessing the view. Conflicts should be handled individually to ensure the consistency of your database.

  To use this mode, you must include the `all_or_nothing` field (set to true) within the main body of the JSON of the request.

The effects of different database operations on the different modes are summarized below:

- **Transaction Mode**: `Non-atomic`
  - **Transaction**: `Insert`
    - **Cause**: Requested document ID already exists
    - **Resolution**: Resubmit with different document ID, or update the existing document
  - **Transaction**: `Update`
    - **Cause**: Revision missing or incorrect
    - **Resolution**: Resubmit with correct revision
- **Transaction Mode**: `All-or-nothing`
  - **Transaction**: `Insert` / `Update`
    - **Cause**: Additional revision inserted
    - **Resolution**: Resolve conflicted revisions

Replication of documents is independent of the type of insert or update. The documents and revisions created during a bulk insert or update are replicated in the same way as any other document. This can mean that if you make use of the all-or-nothing mode the exact list of documents, revisions (and their conflict state) may or may not be replicated to other databases correctly.

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. If you used `all-or-nothing` mode then you will need to manually resolve the conflicted revisions of the document.

  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"
  }
  ```

9.1.10 POST /db/_temp_view

- **Method**: `POST /db/_temp_view`
- **Request**: JSON with the temporary view definition
- **Response**: Temporary view result set
- **Admin Privileges Required**: yes

Creates (and executes) a temporary view based on the view function supplied in the JSON request. For example:

```
POST http://couchdb:5984/recipes/_temp_view
Content-Type: application/json

{
   "map" : "function(doc) { if (doc.value > 9995) { emit(null, doc.value); } }"
}
```

The resulting JSON response is the result from the execution of the temporary view:

```
{
   "total_rows" : 3,
   "rows" : [
      {
         "value" : 9998.41913029012,
         "id" : "05361cc6aa42033878acc1bacb1f39c2",
         "key" : null
      },
      {
         "value" : 9998.94149934853,
         "id" : "1f443f471e5929dd7b252417625ed170",
         "key" : null
      },
      {
         "value" : 9998.01511339154,
         "id" : "1f443f471e5929dd7b252417629c102b",
         "key" : null
      }
   ],
   "offset" : 0
}
```

The arguments also available to standard view requests also apply to temporary views, but the execution of the view may take some time as it relies on being executed at the time of the request. In addition to the time taken, they are also computationally very expensive to produce. You should use a defined view if you want to achieve the best performance.

9.1.11 POST /db/_purge

- **Method**: `POST /db/_purge`
- **Request**: JSON of the document IDs/revisions to be purged
- **Response**: JSON structure with purged documents and purge sequence
- **Admin Privileges Required**: no

Occasionally something into ends up in your database that should never have written to it, like a password or private key. Purge can help you here.

A database purge permanently removes the references to deleted documents from the database. Deleting a document within CouchDB does not actually remove the document from the database, instead, the document is marked as a deleted (and a new revision is created). This is to ensure that deleted documents are replicated to other databases as having been deleted. This also means that you can check the status of a document and identify that the document has been deleted.

Note: Don't use purge as a regular operation, it is designed to be used in exceptional cases.

The purge operation removes the references to the deleted documents from the database. The purging of old documents is not replicated to other databases. If you are replicating between databases and have deleted a large number of documents you should run purge on each database.

Note: Purging documents does not remove the space used by them on disk. To reclaim disk space, you should run a database compact (see *POST /db/_compact*), and compact views (see *POST /db/_compact/design-doc*).

To perform a purge operation you must send a request including the JSON of the document IDs that you want to purge. For example:

```
POST http://couchdb:5984/recipes/_purge
Content-Type: application/json
```

```
{
  "FishStew" : [
    "17-b3eb5ac6fbaef4428d712e66483dcb79"
    ]
}
```

The format of the request must include the document ID and one or more revisions that must be purged.

The response will contain the purge sequence number, and a list of the document IDs and revisions successfully purged.

```
{
   "purged" : {
      "FishStew" : [
         "17-b3eb5ac6fbaef4428d712e66483dcb79"
      ]
   },
   "purge_seq" : 11
}
```

Updating Indexes

The number of purges on a database is tracked using a purge sequence. This is used by the view indexer to optimize the updating of views that contain the purged documents.

When the indexer identifies that the purge sequence on a database has changed, it compares the purge sequence of the database with that stored in the view index. If the difference between the stored sequence and database is sequence is only 1, then the indexer uses a cached list of the most recently purged documents, and then removes these documents from the index individually. This prevents completely rebuilding the index from scratch.

If the difference between the stored sequence number and current database sequence is greater than 1, then the view index is entirely rebuilt. This is an expensive operation as every document in the database must be examined.

9.1.12 `GET /db/_all_docs`

- **Method**: `GET /db/_all_docs`
- **Request**: None
- **Response**: JSON object containing document information, ordered by the document ID
- **Admin Privileges Required**: no
- **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 http://couchdb:5984/recipes/_all_docs
Accept: application/json
```

Returns the following structure:

```
{
   "total_rows" : 18386,
   "rows" : [
      {
         "value" : {
            "rev" : "1-bc0d5aed1e339b1cc1f29578f3220a45"
         },
         "id" : "Aberffrawcake",
         "key" : "Aberffrawcake"
      },
      {
         "value" : {
            "rev" : "3-68a20c89a5e70357c20148f8e82ca331"
         },
         "id" : "Adukiandorangecasserole-microwave",
         "key" : "Adukiandorangecasserole-microwave"
      },
      {
         "value" : {
            "rev" : "3-9b2851ed9b6f655cc4eb087808406c60"
         },
         "id" : "Aioli-garlicmayonnaise",
```

```
            "key" : "Aioli-garlicmayonnaise"
        },
        ...
            ],
    "offset" : 0
}
```

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.

9.1.13 POST /db/_all_docs

- **Method**: `POST /db/_all_docs`
- **Request**: JSON of the document IDs you want included
- **Response**: JSON of the returned view
- **Admin Privileges Required**: no

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 *GET /db/doc* requests.

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

```
POST http://couchdb:5984/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
}
```

9.1.14 POST /db/_missing_revs

- **Method**: `POST /db/_missing_revs`

- **Request**: JSON list of document revisions
- **Response**: JSON of missing revisions
- **Admin Privileges Required**: no

9.1.15 POST /db/_revs_diff

- **Method**: `POST /db/_revs_diff`
- **Request**: JSON list of document revisions
- **Response**: JSON list of differences from supplied document/revision list
- **Admin Privileges Required**: no

9.1.16 GET /db/_security

- **Method**: `GET /db/_security`
- **Request**: None
- **Response**: JSON of the security object
- **Admin Privileges Required**: no

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.

9.1.17 PUT /db/_security

- **Method**: `PUT /db/_security`
- **Request**: JSON specifying the admin and user security for the database
- **Response**: JSON status message
- **Admin Privileges Required**: no

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

```
PUT http://couchdb:5984/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
}
```

9.1.18 GET /db/_revs_limit

- **Method**: `GET /db/_revs_limit`
- **Request**: None
- **Response**: The current revision limit setting
- **Admin Privileges Required**: no

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

For example to get the current limit:

```
GET http://couchdb:5984/recipes/_revs_limit
Content-Type: application/json
```

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

```
1000
```

9.1.19 PUT /db/_revs_limit

- **Method**: `PUT /db/_revs_limit`
- **Request**: A scalar integer of the revision limit setting
- **Response**: Confirmation of setting of the revision limit
- **Admin Privileges Required**: no

Sets the maximum number of document revisions that will be tracked by CouchDB, 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 http://couchdb:5984/recipes/_revs_limit
Content-Type: application/json

100
```

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

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

9.2 Document Methods

The CouchDB API Server Document methods detail how to create, read, update and delete documents within a database.

A list of the available methods and URL paths are 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

9.2.1 POST /db

- **Method**: `POST /db`
- **Request**: JSON of the new document
- **Response**: JSON with the committed document information
- **Admin Privileges Required**: no
- **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 http://couchdb:5984/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 http://couchdb:5984/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

CouchDB supports a number of different UUID generation algorithms for use in situations where a user-specified UUID does not make sense. These can be set simply by *PUT http://couchdb:5984/_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 CouchDB 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 *PUT /db/doc/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 *GET /db/doc/attachment*.

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

9.2.2 GET /db/doc

- **Method**: `GET /db/doc`
- **Request**: None
- **Response**: Returns the JSON for the document
- **Admin Privileges Required**: no
- **Query Arguments**:
  - **Argument**: conflicts
    - **Description**: Returns the conflict tree for the document.
    - **Optional**: yes
    - **Type**: boolean
    - **Default**: false
    - **Supported Values**:
      - **true**: Includes the 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 `FishStew` you would send the following request:

```
GET http://couchdb:5984/recipes/FishStew
Content-Type: application/json
Accept: application/json
```

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

```
{
   "_id" : "FishStew",
   "_rev" : "3-a1a9b39ee3cc39181b796a69cb48521c",
   "servings" : 4,
   "subtitle" : "Delicious with a green salad",
   "title" : "Irish Fish Stew"
}
```

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" : "FishStew",
   "servings" : 4,
   "subtitle" : "Delicious with fresh bread",
   "title" : "Fish Stew"
   "_attachments" : {
      "styling.css" : {
         "stub" : true,
         "content-type" : "text/css",
         "length" : 783426,
         },
   },
}
```

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
    - **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 http://couchdb:5984/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**: CouchDB 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 http://couchdb:5984/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]: CouchDB 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 http://couchdb:5984/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"
}
```

9.2.3 `HEAD /db/doc`

- **Method**: `HEAD /db/doc`
- **Request**: None
- **Response**: None
- **Admin Privileges Required**: no
- **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 http://couchdb:5984/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
```

9.2.4 PUT /db/doc

- **Method**: `PUT /db/doc`
- **Request**: JSON of the new document, or updated version of the existed document
- **Response**: JSON of the document ID and revision
- **Admin Privileges Required**: no
- **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 `FishStew`, you would send the following request:

```
PUT http://couchdb:5984/recipes/FishStew
Content-Type: application/json

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

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

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

Updating an Existing Document

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

```
PUT http://couchdb:5984/recipes/FishStew
Content-Type: application/json

{
  "_rev" : "1-9c65296036141e575d32ba9c034dd3ee",
  "servings" : 4,
  "subtitle" : "Delicious with fresh salad",
  "title" : "Fish Stew"
}
```

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

```
PUT http://couchdb:5984/recipes/FishStew
If-Match: 2-d953b18035b76f2a5b1d1d93f25d3aea
Content-Type: application/json

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

The JSON returned will include the updated revision number:

```
{
   "id" : "FishStew99",
   "ok" : true,
   "rev" : "2-d953b18035b76f2a5b1d1d93f25d3aea"
}
```

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

9.2.5 DELETE /db/doc

- **Method**: `DELETE /db/doc`
- **Request**: None
- **Response**: JSON of the deleted revision
- **Admin Privileges Required**: no
- **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 http://couchdb:5984/recipes/FishStew?rev=3-a1a9b39ee3cc39181b796a69cb48521c
Content-Type: application/json
```

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

```
DELETE http://couchdb:5984/recipes/FishStew
If-Match: 3-a1a9b39ee3cc39181b796a69cb48521c
Content-Type: application/json
```

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

```
{
   "id" : "FishStew",
   "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.

9.2.6 COPY /db/doc

- **Method**: `COPY /db/doc`
- **Request**: None
- **Response**: JSON of the new document and revision
- **Admin Privileges Required**: no
- **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

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

```
COPY http://couchdb:5984/recipes/FishStew
Content-Type: application/json
Destination: IrishFishStew
```

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

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

Copying from a Specific Revision

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

```
COPY http://couchdb:5984/recipes/FishStew?rev=5-acfd32d233f07cea4b4f37daaacc0082
Content-Type: application/json
Destination: IrishFishStew
```

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 http://couchdb:5984/recipes/FishStew
Content-Type: application/json
Destination: IrishFishStew?rev=1-9c65296036141e575d32ba9c034dd3ee
```

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

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

9.2.7 GET /db/doc/attachment

- **Method**: `GET /db/doc/attachment`
- **Request**: None
- **Response**: Returns the attachment data
- **Admin Privileges Required**: no

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.

9.2.8 PUT /db/doc/attachment

- **Method**: `PUT /db/doc/attachment`
- **Request**: Raw document data
- **Response**: JSON document status
- **Admin Privileges Required**: no
- **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 http://couchdb:5984/recipes/FishStew/basic?rev=8-a94cb7e50ded1e06f943be5bfbddf8ca
Content-Length: 10
Content-Type: text/plain

Roast it
```

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

```
PUT http://couchdb:5984/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.

9.2.9 `DELETE /db/doc/attachment`

- **Method**: `DELETE /db/doc/attachment`
- **Request**: None
- **Response**: JSON status
- **Admin Privileges Required**: no
- **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 http://couchdb:5984/recipes/FishStew/basic?rev=9-247bb19a41bfd9bfdaf5ee6e2e05be74
Content-Type: application/json
```

The returned JSON contains the updated revision information:

```
{
   "id" : "FishStew",
   "ok" : true,
   "rev" : "10-561bf6b1e27615cee83d1f48fa65dd3e"
}
```

9.3 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 CouchDB 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

9.3.1 `GET /db/_local/local-doc`

- **Method**: `GET /db/_local/local-doc`
- **Request**: None
- **Response**: JSON of the returned document
- **Admin Privileges Required**: no
- **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 *GET /db/doc*.

9.3.2 `PUT /db/_local/local-doc`

- **Method**: `PUT /db/_local/local-doc`
- **Request**: JSON of the document
- **Response**: JSON with the committed document information
- **Admin Privileges Required**: no
- **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 *PUT /db/doc*.

9.3.3 `DELETE /db/_local/local-doc`

- **Method**: `DELETE /db/_local/local-doc`
- **Request**: None
- **Response**: JSON with the deleted document information
- **Admin Privileges Required**: no
- **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 *DELETE /db/doc*.

9.3.4 `COPY /db/_local/local-doc`

- **Method**: `COPY /db/_local/local-doc`
- **Request**: None
- **Response**: JSON of the copied document
- **Admin Privileges Required**: no
- **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 *COPY /db/doc*.

9.4 Design Document Methods

In CouchDB, design documents provide the main interface for building a CouchDB application. The design document defines the views used to extract information from CouchDB through one or more views. Design documents are created within your CouchDB instance in the same way as you create 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.

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

Design Document API Calls

9.4.1 `GET /db/_design/design-doc`

- **Method**: `GET /db/_design/design-doc`
- **Request**: None
- **Response**: JSON of the existing design document
- **Admin Privileges Required**: no
- **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
    - **Supported Values**:
      - **true**: Includes the revisions
  - **Argument**: revs_info
    - **Description**: Return a list of detailed revision information for the document
    - **Optional**: yes
    - **Type**: boolean
    - **Supported Values**:
      - **true**: Includes the revisions

Returns the specified design document, `design-doc` from the specified `db`. For example, to retrieve the design document `recipes` you would send the following request:

```
GET http://couchdb:5984/recipes/_design/recipes
Content-Type: application/json
```

The returned string will be the JSON of the design document:

```
{
   "_id" : "_design/recipes",
   "_rev" : "5-39f56a392b86bbee57e2138921346406"
   "language" : "javascript",
   "views" : {
      "by_recipe" : {
         "map" : "function(doc) { if (doc.title != null) emit(doc.title, doc) }"
      },
   },
}
```

A list of the revisions can be obtained by using the `revs` query argument, or an extended list of revisions using the `revs_info` query argument. This operates in the same way as for other documents. Fur further examples, see *GET /db/doc*.

9.4.2 PUT /db/_design/design-doc

- **Method**: `PUT /db/_design/design-doc`
- **Request**: JSON of the design document
- **Response**: JSON status
- **Admin Privileges Required**: no

Upload the specified design document, `design-doc`, to the specified database. The design document should follow the definition of a design document, as 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

For more information on writing views, see *GET /db/_design/design-doc/_view/view-name*.

9.4.3 DELETE /db/_design/design-doc

- **Method**: `DELETE /db/_design/design-doc`
- **Request**: None
- **Response**: JSON of deleted design document
- **Admin Privileges Required**: no
- **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 http://couchdb:5984/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",
}
```

9.4.4 COPY /db/_design/design-doc

- **Method**: `COPY /db/_design/design-doc`
- **Request**: None
- **Response**: JSON of the new document and revision
- **Admin Privileges Required**: no

- **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.

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 http://couchdb:5984/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 automatically reconstruct the view indexes. These will be recreated, as with other views, the first time the new view is accessed.

Copying from a Specific Revision

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

```
COPY http://couchdb:5984/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.

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 http://couchdb:5984/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",
}
```

9.4.5 GET /db/_design/design-doc/attachment

- **Method**: `GET /db/_design/design-doc/attachment`
- **Request**: None
- **Response**: Returns the attachment data
- **Admin Privileges Required**: no

Returns the file attachment `attachment` associated with the design document `/_design_/design-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.

9.4.6 PUT /db/_design/design-doc/attachment

- **Method**: `PUT /db/_design/design-doc/attachment`
- **Request**: Raw document data
- **Response**: JSON document status
- **Admin Privileges Required**: no
- **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

Upload the supplied content as an attachment to the specified design document (`/_design/design-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 attacment 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 http://couchdb:5984/recipes/_design/recipes/view.css?rev=7-f7114d4d81124b223283f3e89eee043e
Content-Length: 39
Content-Type: text/plain

div.recipetitle {
font-weight: bold;
}
```

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

```
PUT http://couchdb:5984/recipes/FishStew/basic
If-Match: 7-f7114d4d81124b223283f3e89eee043e
Content-Length: 39
Content-Type: text/plain

div.recipetitle {
font-weight: bold;
}
```

The returned JSON contains the new document information:

```
{
   "id" : "_design/recipes"
   "ok" : true,
   "rev" : "8-cb2b7d94eeac76782a02396ba70dfbf5",
}
```

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

9.4.7 DELETE /db/_design/design-doc/attachment

- **Method**: `DELETE /db/_design/design-doc/attachment`
- **Request**: None
- **Response**: JSON status
- **Admin Privileges Required**: no
- **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 `_design/design-doc`. You must supply the `rev` argument with the current revision to delete the attachment.

For example to delete the attachment `view.css` from the design document `recipes`:

```
DELETE http://couchdb:5984/recipes/_design/recipes/view.css?rev=9-3db559f13a845c7751d407404cdeaa4
```

The returned JSON contains the updated revision information for the parent document:

```
{
   "id" : "_design/recipes"
   "ok" : true,
   "rev" : "10-f3b15bb408961f8dcc3d86c7d3b54c4c",
}
```

9.4.8 GET /db/_design/design-doc/_info

- **Method**: `GET /db/_design/design-doc/_info`
- **Request**: None
- **Response**: JSON of the design document information
- **Admin Privileges Required**: no

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 http://couchdb:5984/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

9.4.9 `GET /db/_design/design-doc/_view/view-name`

- **Method**: `GET /db/_design/design-doc/_view/view-name`
- **Request**: None
- **Response**: JSON of the documents returned by the view
- **Admin Privileges Required**: no
- **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.

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 *GET /db/_changes* for more information.
- Use a monitor with the `update_notification` section of the CouchDB configuration file to monitor for changes to your database, and trigger a view query to force the view to be updated. For more information, see *Update Notifications*.

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:

```
http://couchdb:5984/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 *GET /db*).

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 http://couchdb:5984/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 http://couchdb:5984/recipes/_design/recipes/_view/by_ingredient?startkey=%22carrots%22&endkey=
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 http://couchdb:5984/recipes/_design/recipes/_view/by_ingredient?descending=true&startkey=%22ca
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 http://couchdb:5984/recipes/_design/recipes/_view/by_ingredient?descending=true&startkey=%22eg
Accept: application/json
Content-Type: application/json
```

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.

Using Limits and Skipping Rows

TBC

View Reduction and Grouping

TBC

9.4.10 `POST /db/_design/design-doc/_view/view-name`

- **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
- **Admin Privileges Required**: no
- **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 *GET /db/_design/design-doc/_view/view-name* 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 http://couchdb:5984/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
}
```

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 *GET /db/doc* requests.

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

```
POST http://couchdb:5984/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
}
```

9.4.11 GET /db/_design/design-doc/_show/show-name

- **Method**: `GET /db/_design/design-doc/_show/show-name`
- **Request**: None
- **Response**: Returns the result of the show
- **Admin Privileges Required**: no
- **Query Arguments**:
  - **Argument**: details
    - **Description**: Indicates whether details should be included
    - **Optional**: yes
    - **Type**: string
  - **Argument**: format
    - **Description**: Format of the returned information
    - **Optional**: yes
    - **Type**: string

9.4.12 POST /db/_design/design-doc/_show/show-name/doc

- **Method**: `POST /db/_design/design-doc/_show/show-name`
- **Request**: Custom data
- **Response**: Returns the result of the show
- **Admin Privileges Required**: no

9.4.13 GET /db/_design/design-doc/_list/list-name/other-design-doc/view-nam

- **Method**: `GET /db/_design/design-doc/_list/list-name/other-design-doc/view-name`
- **Request**: TBC
- **Response**: TBC
- **Admin Privileges Required**: no

9.4.14 POST /db/_design/design-doc/_list/list-name/other-design-doc/view-na

- **Method**: `POST /db/_design/design-doc/_list/list-name/other-design-doc/view-name`
- **Request**: TBC
- **Response**: TBC

- **Admin Privileges Required**: no

9.4.15 GET /db/_design/design-doc/_list/list-name/view-name

- **Method**: `GET /db/_design/design-doc/_list/list-name/view-name`
- **Request**: TBC
- **Response**: TBC
- **Admin Privileges Required**: no

9.4.16 POST /db/_design/design-doc/_list/list-name/view-name

- **Method**: `POST /db/_design/design-doc/_list/list-name/view-name`
- **Request**: TBC
- **Response**: TBC
- **Admin Privileges Required**: no

9.4.17 PUT /db/_design/design-doc/_update/updatename/doc

- **Method**: `POST /db/_design/design-doc/_update/updatename/doc`
- **Request**: TBC
- **Response**: TBC
- **Admin Privileges Required**: no

9.4.18 POST /db/_design/design-doc/_update/updatename

- **Method**: `PUT /db/_design/design-doc/_update/updatename/doc`
- **Request**: TBC
- **Response**: TBC
- **Admin Privileges Required**: no

9.4.19 ALL /db/_design/design-doc/_rewrite/rewrite-name/anything

- **Method**: `ALL /db/_design/design-doc/_rewrite/rewrite-name/anything`
- **Request**: TBC
- **Response**: TBC
- **Admin Privileges Required**: no

9.5 Miscellaneous Methods

The CouchDB Miscellaneous interface provides the basic interface to a CouchDB server for obtaining CouchDB information and getting and setting configuration information.

A list of the available methods and URL paths are 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
GET	/_db_updates	A feed of database events
GET	/_log	Return the server log file
POST	/_replicate	Set or cancel replication
POST	/_restart	Restart the server
GET	/_stats	Return server statistics
GET	/_utils	CouchDB administration interface (Futon)
GET	/_uuids	Get generated UUIDs from the server
GET	/favicon.ico	Get the site icon

9.5.1 GET /

- **Method**: `GET /`
- **Request**: None
- **Response**: Welcome message and version
- **Admin Privileges Required**: no
- **Return Codes**:
  - **200**: Request completed successfully.

Accessing the root of a CouchDB instance returns meta information about the instance. The response is a JSON structure containing information about the server, including a welcome message and the version of the server.

```
{
   "couchdb" : "Welcome",
   "version" : "1.0.1"
}
```

9.5.2 GET /_active_tasks

- **Method**: `GET /_active_tasks`
- **Request**: None
- **Response**: List of running tasks, including the task type, name, status and process ID
- **Admin Privileges Required**: yes
- **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:

```
[
   {
    "pid" : "<0.11599.0>",
    "status" : "Copied 0 of 18369 changes (0%)",
    "task" : "recipes",
    "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 operation type, valid values include:

- `Database Compaction`
- `Replication`
- `View Group Compaction`
- `View Group Indexer`

9.5.3 GET /_all_dbs

- **Method**: `GET /_all_dbs`
- **Request**: None
- **Response**: JSON list of DBs
- **Admin Privileges Required**: no
- **Return Codes**:
  - **200**: Request completed successfully.

Returns a list of all the databases in the CouchDB instance. For example:

```
GET http://couchdb:5984/_all_dbs
Accept: application/json
```

The return is a JSON array:

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

9.5.4 GET /_db_updates

- **Method**: `GET /_db_updates`
- **Request**: None
- **Admin Privileges Required**: yes
- **Query ARguments**:
  - **Argument**: feed
    - **Descroption**: Format of the response feed
    - **Optional**: yes
    - **Type**: string
    - **Default**: longpoll
    - **Supported Values**:
      - **longpoll**: Closes the connection after the first event.

        - **continuous**: Send a line of JSON per event. Keeps the socket open until `timeout`.
        - **eventsource**: Like, `continuous`, but sends the events in EventSource format. See http://dev.w3.org/html5/eventsource/ for details,
    - **Argument**: timeout
        - **Descroption**: Number of seconds until CouchDB closes the connection.
        - **Optional**: yes
        - **Type**: numeric
        - **Default**: 60
    - **Argument**: heartbeat
        - **Descroption**: Whether CouchDB will send a newline character (`\n`) on `timeout`.
        - **Optional**: yes
        - **Type**: boolean
        - **Default**: true
- **Return Codes**:
    - **200** Request completed successfully.

Returns a list of all database events in the CouchDB instance.

A database event is one of *created*, *updated*, *deleted*.

For example:

```
GET http://couchdb:5984/_db_events?feed=continuous
Accept: application/json
```

```
{"dbname":"my-database", "type":"created"}
{"dbname":"my-database", "type":"updated"}
{"dbname":"another-database", "type":"created"}
{"dbname":"my-database", "type":"deleted"}
{"dbname":"another-database", "type":"updated"}
```

9.5.5 GET /_log

- **Method**: `GET /_log`
- **Request**: None
- **Response**: Log content
- **Admin Privileges Required**: yes
- **Query Arguments**:
    - **Argument**: bytes
        - **Description**: Bytes to be returned
        - **Optional**: yes
        - **Type**: numeric
        - **Default**: 1000
    - **Argument**: offset
        - **Description**: Offset in bytes where the log tail should be started
        - **Optional**: yes

    * **Type**: numeric
    * **Default**: 0

- **Return Codes**:
  - **200**: Request completed successfully.

Gets the CouchDB log, equivalent to accessing the local log file of the corresponding CouchDB instance.

When you request the log, the response is returned as plain (UTF-8) text, with an HTTP `Content-type` header as `text/plain`.

For example, the request:

```
GET http://couchdb:5984/_log
Accept: */*
```

The raw text is returned:

```
[Wed, 27 Oct 2010 10:49:42 GMT] [info] [<0.23338.2>] 192.168.0.2 - - 'PUT' /authdb 401
[Wed, 27 Oct 2010 11:02:19 GMT] [info] [<0.23428.2>] 192.168.0.116 - - 'GET' /recipes/FishStew 20
[Wed, 27 Oct 2010 11:02:19 GMT] [info] [<0.23428.2>] 192.168.0.116 - - 'GET' /_session 200
[Wed, 27 Oct 2010 11:02:19 GMT] [info] [<0.24199.2>] 192.168.0.116 - - 'GET' / 200
[Wed, 27 Oct 2010 13:03:38 GMT] [info] [<0.24207.2>] 192.168.0.116 - - 'GET' /_log?offset=5 200
```

If you want to pick out specific parts of the log information you can use the `bytes` argument, which specifies the number of bytes to be returned, and `offset`, which specifies where the reading of the log should start, counted back from the end. For example, if you use the following request:

```
GET /_log?bytes=500&offset=2000
```

Reading of the log will start at 2000 bytes from the end of the log, and 500 bytes will be shown.

9.5.6 POST /_replicate

- **Method**: `POST /_replicate`
- **Request**: Replication specification
- **Response**: TBD
- **Admin Privileges Required**: yes
- **Query Arguments**:
  - **Argument**: bytes
    * **Description**: Bytes to be returned
    * **Optional**: yes
    * **Type**: numeric
    * **Default**: 1000
  - **Argument**: offset
    * **Description**: Offset in bytes where the log tail should be started
    * **Optional**: yes
    * **Type**: numeric
    * **Default**: 0
- **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 name or URL
- **target**: Target database name or URL

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 CouchDB 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.

Specifying the Source and Target Database

You must use the URL specification of the CouchDB database if you want to perform replication in either of the following two situations:

- Replication with a remote database (i.e. another instance of CouchDB on the same host, or a different host)
- Replication with a database that requires authentication

For example, to request replication between a database local to the CouchDB instance to which you send the request, and a remote database you might use the following request:

```
POST http://couchdb:5984/_replicate
Content-Type: application/json
Accept: application/json

{
   "source" : "recipes",
   "target" : "http://coucdb-remote:5984/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://couchdb-remote:5984/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://couchdb:5984/_replicate
Content-Type: application/json
Accept: application/json

{
   "create_target" : true
   "source" : "recipes",
   "target" : "http://couchdb-remote:5984/recipes",
}
```

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

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 http://couchdb:5984/_replicate
Content-Type: application/json
Accept: application/json

{
   "source" : "recipes",
   "target" : "recipes-snapshot",
}
```

In the above example, the databases `recipes` and `recipes-snapshot` will be synchronized. These databases are local to the CouchDB instance where the request was made. 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

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 http://couchdb:5984/_replicate
Content-Type: application/json
Accept: application/json

{
   "continuous" : true
   "source" : "recipes",
   "target" : "http://couchdb-remote:5984/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`.

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 http://couchdb:5984/_replicate
Content-Type: application/json
Accept: application/json

{
   "source" : "recipes",
   "target" : "http://couchdb-remote:5984/recipes",
   "create_target" : true,
   "continuous" : true
}
```

Must be canceled using the request:

```
POST http://couchdb:5984/_replicate
Content-Type: application/json
Accept: application/json

{
    "cancel" : true,
    "continuous" : true
    "create_target" : true,
    "source" : "recipes",
    "target" : "http://couchdb-remote:5984/recipes",
}
```

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

9.5.7 POST /_restart

- **Method**: `POST /_restart`
- **Request**: None
- **Response**: JSON status message
- **Admin Privileges Required**: yes
- **HTTP Headers**:
  - **Header**: `Content-Type`
    - **Description**: Request content type
    - **Optional**: no
    - **Value**: *application/json*
- **Return Codes**:
  - **200**: Replication request successfully completed

Restarts the CouchDB instance. You must be authenticated as a user with administration privileges for this to work.

For example:

```
POST http://admin:password@couchdb:5984/_restart
```

The return value (if the server has not already restarted) is a JSON status object indicating that the request has been received:

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

If the server has already restarted, the header may be returned, but no actual data is contained in the response.

9.5.8 GET /_stats

- **Method**: GET /_stats
- **Request**: None
- **Response**: Server statistics
- **Admin Privileges Required**: no
- **Return Codes**:
  - **200**: Request completed successfully.

The _stats method returns a JSON object containing the statistics for the running server. The object is structured with top-level sections collating the statistics for a range of entries, with each individual statistic being easily identified, and the content of each statistic is self-describing. For example, the request time statistics, within the couchdb section are structured as follows:

```
{
   "couchdb" : {
...
      "request_time" : {
         "stddev" : "27.509",
         "min" : "0.333333333333333",
         "max" : "152",
         "current" : "400.976",
         "mean" : "10.837",
         "sum" : "400.976",
         "description" : "length of a request inside CouchDB without MochiWeb"
      },
...
    }
}
```

The fields provide the current, minimum and maximum, and a collection of statistical means and quantities. The quantity in each case is not defined, but the descriptions below provide

The statistics are divided into the following top-level sections:

- couchdb: Describes statistics specific to the internals of CouchDB.

Statistic ID	Description	Unit
auth_cache_hits	Number of authentication cache hits	number
auth_cache_misses	Number of authentication cache misses	number
database_reads	Number of times a document was read from a database	number
database_writes	Number of times a database was changed	number
open_databases	Number of open databases	number
open_os_files	Number of file descriptors CouchDB has open	number
request_time	Length of a request inside CouchDB without MochiWeb	milliseconds

- httpd_request_methods

Statistic ID	Description	Unit
COPY	Number of HTTP COPY requests	number
DELETE	Number of HTTP DELETE requests	number
GET	Number of HTTP GET requests	number
HEAD	Number of HTTP HEAD requests	number
POST	Number of HTTP POST requests	number
PUT	Number of HTTP PUT requests	number

- httpd_status_codes

Statistic ID	Description	Unit
200	Number of HTTP 200 OK responses	number
201	Number of HTTP 201 Created responses	number
202	Number of HTTP 202 Accepted responses	number
301	Number of HTTP 301 Moved Permanently responses	number
304	Number of HTTP 304 Not Modified responses	number
400	Number of HTTP 400 Bad Request responses	number
401	Number of HTTP 401 Unauthorized responses	number
403	Number of HTTP 403 Forbidden responses	number
404	Number of HTTP 404 Not Found responses	number
405	Number of HTTP 405 Method Not Allowed responses	number
409	Number of HTTP 409 Conflict responses	number
412	Number of HTTP 412 Precondition Failed responses	number
500	Number of HTTP 500 Internal Server Error responses	number

- `httpd`

Statistic ID	Description	Unit
bulk_requests	Number of bulk requests	number
clients_requesting_changes	Number of clients for continuous _changes	number
requests	Number of HTTP requests	number
temporary_view_reads	Number of temporary view reads	number
view_reads	Number of view reads	number

You can also access individual statistics by quoting the statistics sections and statistic ID as part of the URL path. For example, to get the `request_time` statistics, you can use:

```
GET /_stats/couchdb/request_time
```

This returns an entire statistics object, as with the full request, but containing only the request individual statistic. Hence, the returned structure is as follows:

```
{
   "couchdb" : {
      "request_time" : {
         "stddev" : 7454.305,
         "min" : 1,
         "max" : 34185,
         "current" : 34697.803,
         "mean" : 1652.276,
         "sum" : 34697.803,
         "description" : "length of a request inside CouchDB without MochiWeb"
      }
   }
}
```

9.5.9 GET /_utils

- **Method**: `GET /_utils`
- **Request**: None
- **Response**: Administration interface
- **Admin Privileges Required**: no

Accesses the built-in Futon administration interface for CouchDB.

9.5.10 GET /_uuids

- **Method**: `GET /_uuids`

- **Request**: None
- **Response**: List of UUIDs
- **Admin Privileges Required**: no
- **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) from the CouchDB instance. 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 http://couchdb:5984/_uuids?count=5
```

Returns:

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

The UUID type is determined by the UUID type setting in the CouchDB configuration. See *PUT /_config/section/key*.

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

```
PUT http://couchdb:5984/_config/uuids/algorithm
Content-Type: application/json
Accept: */*

"random"
```

When obtaining a list of UUIDs:

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

9.5.11 GET /favicon.ico

- **Method**: `GET /favicon.ico`
- **Request**: None
- **Response**: Binary content for the *favicon.ico* site icon
- **Admin Privileges Required**: no
- **Return Codes**:
  - **200**: Request completed successfully.
  - **404**: The requested content could not be found. The returned content will include further information, as a JSON object, if available.

Returns the site icon. The return `Content-Type` header is *`image/x-icon`*, and the content stream is the image data.

9.6 Configuration Methods

The CouchDB API Server Configuration Methods provide an interface to query and update the various configuration values within a running CouchDB instance.

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

Method	Path	Description
GET	/_config	Obtain a list of the entire server configuration
GET	/_config/section	Get all the configuration values for the specified section
GET	/_config/section/key	Get a specific section/configuration value
PUT	/_config/section/key	Set the specified configuration value
DELETE	/_config/section/key	Delete the current setting

9.6.1 GET /_config

- **Method**: `GET /_config`
- **Request**: None
- **Response**: Returns a structure configuration name and value pairs, organized by section
- **Admin Privileges Required**: yes
- **Return Codes**:
  - **200**: Request completed successfully.

Returns the entire CouchDB server configuration as a JSON structure. The structure is organized by different configuration sections, with individual values.

For example, to get the configuration for a server:

```
GET http://couchdb:5984/_config
Accept: application/json
```

The response is the JSON structure:

```
{
   "query_server_config" : {
      "reduce_limit" : "true"
   },
   "couchdb" : {
      "os_process_timeout" : "5000",
      "max_attachment_chunk_size" : "4294967296",
```

```
    "max_document_size" : "4294967296",
    "uri_file" : "/var/lib/couchdb/couch.uri",
    "max_dbs_open" : "100",
    "view_index_dir" : "/var/lib/couchdb",
    "util_driver_dir" : "/usr/lib64/couchdb/erlang/lib/couch-1.0.1/priv/lib",
    "database_dir" : "/var/lib/couchdb",
    "delayed_commits" : "true"
},
"attachments" : {
    "compressible_types" : "text/*, application/javascript, application/json,  application/xml",
    "compression_level" : "8"
},
"uuids" : {
    "algorithm" : "utc_random"
},
"daemons" : {
    "view_manager" : "{couch_view, start_link, []}",
    "auth_cache" : "{couch_auth_cache, start_link, []}",
    "uuids" : "{couch_uuids, start, []}",
    "stats_aggregator" : "{couch_stats_aggregator, start, []}",
    "query_servers" : "{couch_query_servers, start_link, []}",
    "httpd" : "{couch_httpd, start_link, []}",
    "stats_collector" : "{couch_stats_collector, start, []}",
    "db_update_notifier" : "{couch_db_update_notifier_sup, start_link, []}",
    "external_manager" : "{couch_external_manager, start_link, []}"
},
"stats" : {
    "samples" : "[0, 60, 300, 900]",
    "rate" : "1000"
},
"httpd" : {
    "vhost_global_handlers" : "_utils, _uuids, _session, _oauth, _users",
    "secure_rewrites" : "true",
    "authentication_handlers" : "{couch_httpd_oauth, oauth_authentication_handler},
                                 {couch_httpd_auth, cookie_authentication_handler},
                                 {couch_httpd_auth, default_authentication_handler}",
    "port" : "5984",
    "default_handler" : "{couch_httpd_db, handle_request}",
    "allow_jsonp" : "false",
    "bind_address" : "192.168.0.2",
    "max_connections" : "2048"
},
"query_servers" : {
    "javascript" : "/usr/bin/couchjs /usr/share/couchdb/server/main.js"
},
"couch_httpd_auth" : {
    "authentication_db" : "_users",
    "require_valid_user" : "false",
    "authentication_redirect" : "/_utils/session.html",
    "timeout" : "600",
    "auth_cache_size" : "50"
},
"httpd_db_handlers" : {
    "_design" : "{couch_httpd_db, handle_design_req}",
    "_compact" : "{couch_httpd_db, handle_compact_req}",
    "_view_cleanup" : "{couch_httpd_db, handle_view_cleanup_req}",
    "_temp_view" : "{couch_httpd_view, handle_temp_view_req}",
    "_changes" : "{couch_httpd_db, handle_changes_req}"
},
"replicator" : {
    "max_http_sessions" : "10",
    "max_http_pipeline_size" : "10"
},
```

```
    "log" : {
        "include_sasl" : "true",
        "level" : "info",
        "file" : "/var/log/couchdb/couch.log"
    },
    "httpd_design_handlers" : {
        "_update" : "{couch_httpd_show, handle_doc_update_req}",
        "_show" : "{couch_httpd_show, handle_doc_show_req}",
        "_info" : "{couch_httpd_db,   handle_design_info_req}",
        "_list" : "{couch_httpd_show, handle_view_list_req}",
        "_view" : "{couch_httpd_view, handle_view_req}",
        "_rewrite" : "{couch_httpd_rewrite, handle_rewrite_req}"
    },
    "httpd_global_handlers" : {
        "_replicate" : "{couch_httpd_misc_handlers, handle_replicate_req}",
        "/" : "{couch_httpd_misc_handlers, handle_welcome_req, <<\"Welcome\">>}",
        "_config" : "{couch_httpd_misc_handlers, handle_config_req}",
        "_utils" : "{couch_httpd_misc_handlers, handle_utils_dir_req, \"/usr/share/couchdb/www\"}",
        "_active_tasks" : "{couch_httpd_misc_handlers, handle_task_status_req}",
        "_session" : "{couch_httpd_auth, handle_session_req}",
        "_log" : "{couch_httpd_misc_handlers, handle_log_req}",
        "favicon.ico" : "{couch_httpd_misc_handlers, handle_favicon_req, \"/usr/share/couchdb/www\"
        "_all_dbs" : "{couch_httpd_misc_handlers, handle_all_dbs_req}",
        "_oauth" : "{couch_httpd_oauth, handle_oauth_req}",
        "_restart" : "{couch_httpd_misc_handlers, handle_restart_req}",
        "_uuids" : "{couch_httpd_misc_handlers, handle_uuids_req}",
        "_stats" : "{couch_httpd_stats_handlers, handle_stats_req}"
    }
}
```

9.6.2 GET /_config/section

- **Method**: `GET /_config/section`
- **Request**: None
- **Response**: All the configuration values within a specified section
- **Admin Privileges Required**: yes
- **Return Codes**:
  - **200**: Request completed successfully.

Gets the configuration structure for a single section. For example, to retrieve the CouchDB configuration section values:

```
GET http://couchdb:5984/_config/couchdb
Accept: application/json
```

The returned JSON contains just the configuration values for this section:

```
{
    "os_process_timeout" : "5000",
    "max_attachment_chunk_size" : "4294967296",
    "max_document_size" : "4294967296",
    "uri_file" : "/var/lib/couchdb/couch.uri",
    "max_dbs_open" : "100",
    "view_index_dir" : "/var/lib/couchdb",
    "util_driver_dir" : "/usr/lib64/couchdb/erlang/lib/couch-1.0.1/priv/lib",
    "database_dir" : "/var/lib/couchdb",
    "delayed_commits" : "true"
}
```

9.6.3 GET /_config/section/key

- **Method**: `GET /_config/section/key`
- **Request**: None
- **Response**: Value of the specified key/section
- **Admin Privileges Required**: yes
- **Return Codes**:
  - **200**: Request completed successfully.

Gets a single configuration value from within a specific configuration section. For example, to obtain the current log level:

```
GET http://couchdb:5984/_config/log/level
Accept: application/json
```

Returns the string of the log level:

```
"info"
```

Note: The returned value will be the JSON of the value, which may be a string or numeric value, or an array or object. Some client environments may not parse simple strings or numeric values as valid JSON.

9.6.4 PUT /_config/section/key

- **Method**: `PUT /_config/section/key`
- **Request**: Value structure
- **Response**: Previous value
- **Admin Privileges Required**: yes
- **Return Codes**:
  - **200**: Configuration option updated successfully
  - **500**: Error setting configuration

Updates a configuration value. The new value should be supplied in the request body in the corresponding JSON format. For example, if you are setting a string value, you must supply a valid JSON string.

For example, to set the function used to generate UUIDs by the `GET /_uuids` API call to use the `utc_random` generator:

```
PUT http://couchdb:5984/_config/uuids/algorithm
Content-Type: application/json

"utc_random"
```

The return value will be empty, with the response code indicating the success or failure of the configuration setting.

9.6.5 DELETE /_config/section/key

- **Method**: `DELETE /_config/section/key`
- **Request**: None
- **Response**: Previous value
- **Admin Privileges Required**: yes

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

Deletes a configuration value. The returned JSON will be the value of the configuration parameter before it was deleted. For example, to delete the UUID parameter:

```
DELETE http://couchdb:5984/_config/uuids/algorithm
Content-Type: application/json
```

The returned value is the last configured UUID function:

```
"random"
```

9.7 Authentication Methods

The CouchDB Authentication methods provide an interface for obtaining session and authorization data.

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

Method	Path	Description
GET	/_oauth/access_token	TBC
GET	/_oauth/authorize	TBC
POST	/_oauth/authorize	TBC
GET	/_oauth/request_token	TBC
GET	/_session	Returns cookie based login user information
POST	/_session	Do cookie based user login
DELETE	/_session	Logout cookie based user

CHAPTER

TEN

JSON STRUCTURE REFERENCE

The following appendix provides a quick reference to all the JSON structures that you can supply to CouchDB, or get in return to requests.

10.1 All Database Documents

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

10.2 Bulk Document Response

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

10.3 Bulk Documents

Field	Description
all_or_nothing (optional)	Sets the database commit mode to use all-or-nothing semantics
docs [array]	Bulk Documents Document
_id (optional)	Document ID
_rev (optional)	Revision ID (when updating an existing document)
_deleted (optional)	Whether the document should be deleted

10.4 Changes information for a database

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

10.5 CouchDB Document

Field	Description
_id (optional)	Document ID
_rev (optional)	Revision ID (when updating an existing document)

10.6 CouchDB Error Status

Field	Description
id	Document ID
error	Error type
reason	Error string with extended reason

10.7 CouchDB database information object

Field	Description
db_name	The name of the database.
committed_update_seq	The number of committed update.
doc_count	A count of the documents in the specified database.
doc_del_count	Number of deleted documents
compact_running	Set to true if the database compaction routine is operating on this 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.
instance_start_time	Timestamp of when the database was opened, expressed in microseconds since the epoch.
purge_seq	The number of purge operations on the database.
update_seq	The current number of updates to the database.

10.8 Design Document

Field	Description
_id	Design Document ID
_rev	Design Document Revision
views	View
viewname	View Definition
map	Map Function for View
reduce (optional)	Reduce Function for View

10.9 Design Document Information

Field	Description
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

10.10 Document with Attachments

Field	Description
_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

10.11 List of Active Tasks

Field	Description
tasks [array]	Active Task
pid	Process ID
status	Task status message
task	Task name
type	Operation Type

10.12 Replication Settings

Field	Description
source	Source database name or URL
target	Target database name or URL
create_target (optional)	Creates the target database
continuous (optional)	Configure the replication to be continuous
cancel (optional)	Cancels the replication
doc_ids (optional)	Array of document IDs to be synchronized
proxy (optional)	Address of a proxy server through which replication should occur
since_seq (optional)	Sequence from which the replication should start
filter (optional)	name of the filter function in the form of ddoc/myfilter
query_params (optional)	query parameter that are passed to the filter function; value should be a document containing parameters as members

10.13 Replication Status

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

10.14 Request object

Field	Description
body	Request body data as *string*. If request method is *GET* method contains this field contains `"undefined"` value, while if *DELETE* or *HEAD* value is `""` (empty string)
cookie	Cookies *object*.
form	Form data *object*. Contains decoded body as key-value pairs if *Content-Type* header was `application/x-www-form-urlencoded`.
headers	Request headers *object*.
id	Requested document id *string* if it was specified or `null` otherwise.
info	*Database information*
method	Request method as *string* or *array*. String value is method is one of: *HEAD*, *GET*, *POST*, *PUT*, *DELETE*, *OPTIONS*, and *TRACE*, otherwise it will be represented as array of char codes.
path	List of requested path sections.
peer	Request source IP address.
query	URL query parameters *object*. Note that multiple keys not supported and last key value suppress others.
requested_path	List of actual requested path section.
raw_path	Raw requested path *string*.
secObj	*Security Object*.
userCtx	*User Context Object*.
uuid	Generated UUID by specified algorithm in config file.

```
{
    "body": "undefined",
    "cookie": {
        "AuthSession": "cm9vdDo1MDZBRjQzRjrfcuikzPRfAn-EA37FmjyfM8G8Lw",
        "m": "3234"
    },
    "form": {},
    "headers": {
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
        "Accept-Charset": "ISO-8859-1,utf-8;q=0.7,*;q=0.3",
        "Accept-Encoding": "gzip,deflate,sdch",
        "Accept-Language": "en-US,en;q=0.8",
        "Connection": "keep-alive",
```

```
        "Cookie": "m=3234:t|3247:t|6493:t|6967:t|34e2:|18c3:t|2c69:t|5acb:t|ca3:t|c01:t|5e55:t|77
        "Host": "127.0.0.1:5984",
        "User-Agent": "Mozilla/5.0 (Windows NT 5.2) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/
    },
    "id": "foo",
    "info": {
        "committed_update_seq": 2701412,
        "compact_running": false,
        "data_size": 7580843252,
        "db_name": "mailbox",
        "disk_format_version": 6,
        "disk_size": 14325313673,
        "doc_count": 2262757,
        "doc_del_count": 560,
        "instance_start_time": "1347601025628957",
        "purge_seq": 0,
        "update_seq": 2701412
    },
    "method": "GET",
    "path": [
        "mailbox",
        "_design",
        "request",
        "_show",
        "dump",
        "foo"
    ],
    "peer": "127.0.0.1",
    "query": {},
    "raw_path": "/mailbox/_design/request/_show/dump/foo",
    "requested_path": [
        "mailbox",
        "_design",
        "request",
        "_show",
        "dump",
        "foo"
    ],
    "secObj": {
        "admins": {
            "names": [
                "Bob"
            ],
            "roles": []
        },
        "members": {
            "names": [
                "Mike",
                "Alice"
            ],
            "roles": []
        }
    },
    "userCtx": {
        "db": "mailbox",
        "name": "Mike",
        "roles": [
            "user"
        ]
    },
    "uuid": "3184f9d1ea934e1f81a24c71bde5c168"
}
```

10.15 Response object

Field	Description
code	HTTP status code *number*.
json	JSON encodable *object*. Implicitly sets *Content-Type* header as `application/json`.
body	Raw response text *string*. Implicitly sets *Content-Type* header as `text/html; charset=utf-8`.
base64	Base64 encoded *string*. Implicitly sets *Content-Type* header as `application/binary`.
headers	Response headers *object*. *Content-Type* header from this object overrides any implicitly assigned one.
stop	*boolean* signal to stop iteration over view result rows (for list functions only)

Warning: `body`, `base64` and `json` object keys are overlaps each other and the last wins. Since most realizations of key-value objects doesn't preserve key order mixing them may create confusing situation. Try to use only one of them.

Note: Any custom property makes CouchDB raise internal exception. Also *Response object* could be a simple string value which would be implicitly wrapped into `{"body": ...}` object.

10.16 Returned CouchDB Document with Detailed Revision Info

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

10.17 Returned CouchDB Document with Revision Info

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

10.18 Returned Document with Attachments

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

10.19 Security Object

Field	Description
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

```
{
    "admins": {
        "names": [
            "Bob"
        ],
        "roles": []
    },
    "members": {
        "names": [
            "Mike",
            "Alice"
        ],
        "roles": []
    }
  }
```

10.20 User Context Object

Field	Description
db	Database name in context of provided operation.
name	User name.
roles	List of user roles.

```
{
    "db": "mailbox",
    "name": null,
    "roles": [
        "_admin"
    ]
}
```

10.21 View Head Information

Field	Description
total_rows	Number of documents in the view
offset	Offset where the document list started

```
{
    "total_rows": 42,
    "offset": 3
}
```

10.22 Number Handling

Any numbers defined in JSON that contain a decimal point or exponent will be passed through the Erlang VM's idea of the "double" data type. Any numbers that are used in views will pass through the views idea of a number (the common JavaScript case means even integers pass through a double due to JavaScript's definition of a number).

Consider this document that we write to CouchDB:

```
{
  "_id":"30b3b38cdbd9e3a587de9b8122000cff",
  "number": 1.1
}
```

Now let's read that document back from CouchDB:

```
{
  "_id":"30b3b38cdbd9e3a587de9b8122000cff",
  "_rev":"1-f065cee7c3fd93aa50f6c97acde93030",
  "number":1.1000000000000000888
}
```

What happens is CouchDB is changing the textual representation of the result of decoding what it was given into some numerical format. In most cases this is an IEEE 754 double precision floating point number which is exactly what almost all other languages use as well.

What CouchDB does a bit differently than other languages is that it does not attempt to pretty print the resulting output to use the shortest number of characters. For instance, this is why we have this relationship:

```
ejson:encode(ejson:decode(<<"1.1">>)).
<<"1.1000000000000000888">>
```

What can be confusing here is that internally those two formats decode into the same IEEE-754 representation. And more importantly, it will decode into a fairly close representation when passed through all major parsers that I know about.

While we've only been discussing cases where the textual representation changes, another important case is when an input value is contains more precision than can actually represented in a double. (You could argue that this case is actually "losing" data if you don't accept that numbers are stored in doubles).

Here's a log for a couple of the more common JSON libraries I happen to have on my machine:

Spidermonkey:

```
$ js -h 2>&1 | head -n 1
JavaScript-C 1.8.5 2011-03-31
$ js
js> JSON.stringify(JSON.parse("1.01234567890123456789012345678901234567890"))
"1.0123456789012346"
js> var f = JSON.stringify(JSON.parse("1.01234567890123456789012345678901234567890"))
js> JSON.stringify(JSON.parse(f))
"1.0123456789012346"
```

Node:

```
$ node -v
v0.6.15
$ node
JSON.stringify(JSON.parse("1.01234567890123456789012345678901234567890"))
'1.0123456789012346'
var f = JSON.stringify(JSON.parse("1.01234567890123456789012345678901234567890"))
undefined
JSON.stringify(JSON.parse(f))
'1.0123456789012346'
```

Python:

```
$ python
Python 2.7.2 (default, Jun 20 2012, 16:23:33)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
import json
json.dumps(json.loads("1.01234567890123456789012345678901234567890"))
'1.0123456789012346'
f = json.dumps(json.loads("1.01234567890123456789012345678901234567890"))
json.dumps(json.loads(f))
'1.0123456789012346'
```

Ruby:

```
$ irb --version
irb 0.9.5(05/04/13)
require 'JSON'
=> true
JSON.dump(JSON.load("[1.01234567890123456789012345678901234567890]"))
=> "[1.01234567890123]"
f = JSON.dump(JSON.load("[1.01234567890123456789012345678901234567890]"))
=> "[1.01234567890123]"
JSON.dump(JSON.load(f))
=> "[1.01234567890123]"
```

Note: A small aside on Ruby, it requires a top level object or array, so I just wrapped the value. Should be obvious it doesn't affect the result of parsing the number though.

Ejson (CouchDB's current parser) at CouchDB sha 168a663b:

```
$ ./utils/run -i
Erlang R14B04 (erts-5.8.5) [source] [64-bit] [smp:2:2] [rq:2]
[async-threads:4] [hipe] [kernel-poll:true]

Eshell V5.8.5  (abort with ^G)
1> ejson:encode(ejson:decode(<<"1.01234567890123456789012345678901234567890">>)).
<<"1.0123456789012346135">>
2> F = ejson:encode(ejson:decode(<<"1.01234567890123456789012345678901234567890">>)).
<<"1.0123456789012346135">>
3> ejson:encode(ejson:decode(F)).
<<"1.0123456789012346135">>
```

As you can see they all pretty much behave the same except for Ruby actually does appear to be losing some precision over the other libraries.

The astute observer will notice that ejson (the CouchDB JSON library) reported an extra three digits. While its tempting to think that this is due to some internal difference, its just a more specific case of the 1.1 input as described above.

The important point to realize here is that a double can only hold a finite number of values. What we're doing here is generating a string that when passed through the "standard" floating point parsing algorithms (ie, strtod) will result in the same bit pattern in memory as we started with. Or, slightly different, the bytes in a JSON serialized number are chosen such that they refer to a single specific value that a double can represent.

The important point to understand is that we're mapping from one infinite set onto a finite set. An easy way to see this is by reflecting on this:

```
1.0 == 1.00 == 1.000 = 1.(infinite zeroes)
```

Obviously a computer can't hold infinite bytes so we have to decimate our infinitely sized set to a finite set that can be represented concisely.

The game that other JSON libraries are playing is merely:

"How few characters do I have to use to select this specific value for a double"

And that game has lots and lots of subtle details that are difficult to duplicate in C without a significant amount of effort (it took Python over a year to get it sorted with their fancy build systems that automatically run on a number of different architectures).

Hopefully we've shown that CouchDB is not doing anything "funky" by changing input. Its behaving the same as any other common JSON library does, its just not pretty printing its output.

On the other hand, if you actually are in a position where an IEEE-754 double is not a satisfactory datatype for your numbers, then the answer as has been stated is to not pass your numbers through this representation. In JSON this is accomplished by encoding them as a string or by using integer types (although integer types can still bite you if you use a platform that has a different integer representation than normal, ie, JavaScript).

Also, if anyone is really interested in changing this behavior, I'm all ears for contributions to jiffy (which is theoretically going to replace ejson when I get around to updating the build system). The places I've looked for inspiration are TCL and Python. If you know a decent implementation of this float printing algorithm give me a holler.

CHAPTER

ELEVEN

CONFIGURATION REFERENCE

11.1 Configuration Groups

Section	Description
attachments	Attachment options
couchdb	CouchDB specific options
couch_httpd_auth	HTTPD Authentication options
daemons	Daemons and background processes
httpd	HTTPD Server options
httpd_db_handlers	Database Operation handlers
httpd_design_handlers	Handlers for design document operations
httpd_global_handlers	Handlers for global operations
log	Logging options
query_servers	Query Server options
query_server_config	Query server options
replicator	Replicator Options
ssl	SSL (Secure Sockets Layer) Options
stats	Statistics options
uuids	UUID generation options
cors	Cross Origin Resource Sharing settings

11.2 attachments Configuration Options

Option	Description
compressible_types	compressible_types
compression_level	compression_level

11.3 couchdb Configuration Options

Option	Description
database_dir	database_dir
delayed_commits	delayed_commits
max_attachment_chunk_size	max_attachment_chunk_size
max_dbs_open	max_dbs_open
max_document_size	max_document_size
os_process_timeout	os_process_timeout
uri_file	uri_file
util_driver_dir	util_driver_dir
view_index_dir	view_index_dir

11.4 daemons Configuration Options

Option	Description
auth_cache	auth_cache
db_update_notifier	db_update_notifier
external_manager	external_manager
httpd	httpd
httpsd	Enabled HTTPS service
query_servers	query_servers
stats_aggregator	stats_aggregator
stats_collector	stats_collector
uuids	uuids
view_manager	view_manager

11.5 httpd_db_handlers Configuration Options

Option	Description
_changes	_changes
_compact	_compact
_design	_design
_temp_view	_temp_view
_view_cleanup	_view_cleanup

11.6 couch_httpd_auth Configuration Options

Option	Description
auth_cache_size	auth_cache_size
authentication_db	authentication_db
authentication_redirect	authentication_redirect
require_valid_user	require_valid_user
timeout	timeout
iterations	Password key derivation iterations
users_db_public	Allow all users to view user documents
public_fields	World-viewable user document fields

Note: Using the *public_fields* whitelist for user document properties requires setting the *users_db_public* option to *true* (the latter option has no other purpose).

11.7 httpd Configuration Options

Option	Description
allow_jsonp	allow_jsonp
authentication_handlers	authentication_handlers
bind_address	bind_address
default_handler	default_handler
max_connections	max_connections
nodelay	Enable TCP_NODELAY
port	port
secure_rewrites	secure_rewrites
vhost_global_handlers	vhost_global_handlers
enable_cors	enables CORS functionality when true

11.8 httpd_design_handlers Configuration Options

Option	Description
_info	_info
_list	_list
_rewrite	_rewrite
_show	_show
_update	_update
_view	_view

11.9 httpd_global_handlers Configuration Options

Option	Description
/	/
_active_tasks	_active_tasks
_all_dbs	_all_dbs
_config	_config
_log	_log
_oauth	_oauth
_replicate	_replicate
_restart	_restart
_session	_session
_stats	_stats
_utils	_utils
_uuids	_uuids
favicon.ico	favicon.ico

11.10 log Configuration Options

Option	Description
file	file
include_sasl	include_sasl
level	level

11.11 query_servers Configuration Options

Option	Description
javascript	javascript

11.12 query_server_config Configuration Options

Option	Description
reduce_limit	reduce_limit

11.13 replicator Configuration Options

Option	Description
max_http_pipeline_size	max_http_pipeline_size
max_http_sessions	max_http_sessions

11.14 stats Configuration Options

Option	Description
rate	rate
samples	samples

11.15 uuids Configuration Options

Option	Description
algorithm	algorithm

11.16 cors Configuration Options

Option	Description
origins	List of origins, separated by a comma (protocol, host, optional port)
methods	accepted HTTP methods
credentials	*true* sends additional header Access-Control-Allow-Credentials=true

Note that *credentials=true* and *origins=** are mutually exclusive.

11.17 cors vhost Configuration

The same configuration options for *cors* overall may be applied to an individual vhost, within a specific section header, for *example.com* the appropriate section would be: *[cors:http://example.com]*

CHAPTER

TWELVE

CONTRIBUTING TO THIS DOCUMENTATION

The documentation lives in the CouchDB source tree. We'll start by forking and closing the CouchDB GitHub mirror. That will allow us to send the contribution to CouchDB with a pull request.

If you don't have a GitHub account yet, it is a good time to get one, they are free. If you don't want to use GitHub, there are alternate ways to contributing back, that we'll cover next time.

Go to https://github.com/apache/couchdb and click the "fork" button in the top right. This will create a fork of CouchDB in your GitHub account. Mine is *janl*, so my fork lives at https://github.com/janl/couchdb. In the header, it tells me me my "GitHub Clone URL". We need to copy that and start a terminal:

```
$ git clone https://github.com/janl/couchdb.git
$ cd couchdb
$ subl .
```

I'm opening the whole CouchDB source tree in my favourite editor. It gives me the usual directory listing:

```
.git/
.gitignore
.mailmap
.travis.yml
AUTHORS
BUGS
CHANGES
DEVELOPERS
INSTALL
INSTALL.Unix
INSTALL.Windows
LICENSE
Makefile.am
NEWS
NOTICE
README
THANKS.in
acinclude.m4.in
bin/
bootstrap
build-aux/
configure.ac
etc/
license.skip
share/
src/
test/
utils/
var/
```

The documentation sources live in *share/doc/src*, you can safely ignore all the other files and directories.

First we should determine where we want to document this inside the documentation. We can look through http://docs.couchdb.org/en/latest/ for inspiration. The JSON Structure Reference looks like a fine place to write this up.

The current state includes mostly tables describing the JSON structure (after all, that's the title of this chapter), but some prose about the number representation can't hurt. For future reference, since the topic in the thread includes views and different encoding in views (as opposed to the storage engine), we should remember to make a note in the views documentation as well, but we'll leave this for later.

Let's try and find the source file that builds the file http://docs.couchdb.org/en/latest/json-structure.html – we are in luck, under *share/doc/src* we find the file *json-structure.rst*. That looks promising. *.rst* stands for ReStructured Text (see http://thomas-cokelaer.info/tutorials/sphinx/rest_syntax.html for a markup reference), which is an ascii format for writing documents, documentation in this case. Let's have a look and open it.

We see ascii tables with some additional formatting, all looking like the final HTML. So far so easy. For now, let's just add to the bottom of this. We can worry about organising this better later.

We start by adding a new headline:

```
Number Handling
===============
```

Now we paste in the rest of the main email of the thread. It is mostly text, but it includes some code listings. Let's mark them up. We'll turn:

```
ejson:encode(ejson:decode(<<"1.1">>)).
<<"1.1000000000000000888">>
```

Into:

```
.. code-block:: erlang

  ejson:encode(ejson:decode(<<"1.1">>)).
  <<"1.1000000000000000888">>
```

And we follow along with the other code samples. We turn:

```
Spidermonkey

$ js -h 2>&1 | head -n 1
JavaScript-C 1.8.5 2011-03-31
$ js
js> JSON.stringify(JSON.parse("1.0123456789012345678901234567890123456789 0"))
"1.0123456789012346"
js> var f = JSON.stringify(JSON.parse("1.0123456789012345678901234567890123456789 0"))
js> JSON.stringify(JSON.parse(f))
"1.0123456789012346"
```

into:

```
Spidermonkey::

    $ js -h 2>&1 | head -n 1
    JavaScript-C 1.8.5 2011-03-31
    $ js
    js> JSON.stringify(JSON.parse("1.0123456789012345678901234567890123456789 0"))
    "1.0123456789012346"
    js> var f = JSON.stringify(JSON.parse("1.0123456789012345678901234567890123456789 0"))
    js> JSON.stringify(JSON.parse(f))
    "1.0123456789012346"
```

And then follow all the other ones.

I cleaned up the text a little but to make it sound more like a documentation entry as opposed to a post on a mailing list.

The next step would be to validate that we got all the markup right. I'll leave this for later. For now we'll contribute our change back to CouchDB.

First, we commit our changes:

```
$ > git commit -am 'document number encoding'
[master a84b2cf] document number encoding
1 file changed, 199 insertions(+)
```

Then we push the commit to our CouchDB fork:

```
$ git push origin master
```

Next, we go back to our GitHub page https://github.com/janl/couchdb and click the "Pull Request" button. Fill in the description with something useful and hit the "Send Pull Request" button.

And we're done!

CHAPTER

THIRTEEN

RELEASE HISTORY

- 1.4.x Branch
- 1.3.x Branch
- 1.2.x Branch
- 1.1.x Branch
- 1.0.x Branch
- 0.11.x Branch
- 0.10.x Branch
- 0.9.x Branch
- 0.8.x Branch

13.1 1.4.x Branch

- Upgrade Notes
- Version 1.4.0

13.1.1 Upgrade Notes

We now support Erlang/OTP R16B and R16B01; the minimum required version is R14B.

User document role values must now be strings. Other types of values will be refused when saving the user document.

13.1.2 Version 1.4.0

- COUCHDB-1684: Support for server-wide changes feed reporting on creation, updates and deletion of databases. #917d8988
- COUCHDB-1139: it's possible to apply *list* functions to `_all_docs` view. #54fd258e
- Automatic loading of CouchDB plugins. #3fab6bb5
- COUCHDB-1634: Reduce PBKDF2 work factor. #f726bc4d
- Allow storing pre-hashed admin passwords via *_config* API. #c98ba561
- COUCHDB-1772: Prevent invalid JSON output when using *all_or_nothing* *_bulk_docs* API. #dfd39d57
- Add a *configurable whitelist* of user document properties. #8d7ab8b1
- COUCHDB-1852: Support Last-Event-ID header in EventSource changes feeds. #dfd2199a

- Much improved documentation, including an *expanded description* of *validate_doc_update* functions (commit:*ef9ac469*) and a description of how CouchDB handles JSON *number values* (#bbd93f77).
- COUCHDB-1632: Ignore epilogues in multipart/related MIME attachments. #2b4ab67a
- Split up replicator_db tests into multiple independent tests.

13.2 1.3.x Branch

- Upgrade Notes
- Version 1.3.1
- Version 1.3.0

13.2.1 Upgrade Notes

You can upgrade your existing CouchDB 1.0.x installation to 1.3.0 without any specific steps or migration. When you run CouchDB, the existing data and index files will be opened and used as normal.

The first time you run a compaction routine on your database within 1.3.0, the data structure and indexes will be updated to the new version of the CouchDB database format that can only be read by CouchDB 1.3.0 and later. This step is not reversible. Once the data files have been updated and migrated to the new version the data files will no longer work with a CouchDB 1.0.x release.

> **Warning:** If you want to retain support for opening the data files in CouchDB 1.0.x you must back up your data files before performing the upgrade and compaction process.

13.2.2 Version 1.3.1

Replicator

- COUCHDB-1788: Tolerate missing source and target fields in _replicator docs. #869f42e2

Log System

- Don't log about missing .compact files. #06f1a8dc
- COUCHDB-1794: Fix bug in WARN level logging from 1.3.0.

View Server

- COUCHDB-1792: Fix the -S option to couchjs to increase memory limits. #cfaa66cd

Miscellaneous

- Improve documentation: better structure, improve language, less duplication.
- COUCHDB-1784: Improvements to test suite and VPATH build system. #01afaa4f

13.2.3 Version 1.3.0

Database core

- COUCHDB-1512: Validate bind address before assignment. #09ead8a0
- Restore `max_document_size` protection. #bf1eb135

Documentation

- COUCHDB-1523: Import CouchBase documentation and convert them into Sphinx docs

Futon

- COUCHDB-1470: Futon raises popup on attempt to navigate to missed/deleted document. #5da40eef
- COUCHDB-1383: Futon view editor won't allow you to save original view after saving a revision. #ce48342
- COUCHDB-627: Support all timezones. #b1a049bb
- COUCHDB-509: Added view request duration to Futon. #2d2c7d1e
- COUCHDB-1473, COUCHDB-1472: Disable buttons for actions that the user doesn't have permissions to. #7156254d

HTTP Interface

- COUCHDB-431: Introduce experimental *CORS support*. #b90e4021
- COUCHDB-1537: Include user name in show/list *ETags*. #ac320479
- COUCHDB-1511: CouchDB checks *roles* field for *_users* database documents with more care. #41205000
- COUCHDB-1502: Allow users to delete own _users doc. #f0d6f19bc8
- COUCHDB-1501: *Changes feed* now can take special parameter `since=now` to emit changes since current point of time. #3bbb2612
- COUCHDB-1442: No longer rewrites the *X-CouchDB-Requested-Path* during recursive calls to the rewriter. #56744f2f
- COUCHDB-1441: Limit recursion depth in the URL rewriter. Defaults to a maximum of 100 invocations but is configurable. #d076976c
- COUCHDB-1381: Add jquery.couch support for Windows 8 Metro apps. #dfc5d37c
- COUCHDB-1337: Use MD5 for attachment ETag header value. #6d912c9f
- COUCHDB-1321: Variables in rewrite rules breaks OAuth authentication. #c307ba95
- COUCHDB-1285: Allow configuration of vendor and modules version in CouchDB welcome message. #3c24a94d
- COUCHDB-1277: Better query parameter support and code clarity: #7e3c69ba
  - Responses to documents created/modified via form data *POST* to /db/doc or copied with *COPY* should now include *Location* header.
  - Form data POST to /db/doc now includes an *ETag* response header.
  - `?batch=ok` is now supported for *COPY* and *POST* /db/doc updates.
  - `?new_edits=false` is now supported for more operations.
- COUCHDB-1210: Files starting with underscore can be attached and updated now. #05858792

- COUCHDB-1097: Allow *OPTIONS* request to shows and lists functions. #9f53704a
- COUCHDB-1026: Database names are encoded with respect of special characters in the rewriter now. #272d6415
- COUCHDB-986: Added Server-Sent Events protocol to db changes API. See http://www.w3.org/TR/eventsource/ for details. #093d2aa6
- COUCHDB-887: Fix `bytes` and `offset` parameters semantic for *_log* resource (explanation) #ad700014
- COUCHDB-764, COUCHDB-514, COUCHDB-430: Fix sending HTTP headers from `_list` function, #2a74f88375
- Send a 202 response for *_restart*. #b213e16f
- Make password hashing synchronous when using the /_config/admins API. #08071a80
- Fix *_session* for IE7.
- Return `X-Couch-Id` header if doc is created, #98515bf0b9
- Allow any 2xx code to indicate success, #0d50103cfd
- Restore 400 error for empty PUT, #2057b895
- Add support to serve single file with CouchDB, #2774531ff2
- Support auth cookies with `:` characters, #d9566c831d

Log System

- COUCHDB-1380: Minor fixes for logrotate support.
- Improve file I/O error logging and handling, #4b6475da
- Module Level Logging, #b58f069167
- Log 5xx responses at error level, #e896b0b7
- Log problems opening database at ERROR level except for auto-created system dbs, #41667642f7

Replicator

- COUCHDB-1557: Upgrade some code to use BIFs bring good improvements for replication.
- COUCHDB-1363: Fix rarely occurred, but still race condition in changes feed if a quick burst of changes happens while replication is starting the replication can go stale. #573a7bb9
- COUCHDB-1323: Replicator now acts as standalone application. #f913ca6e
- COUCHDB-1259: Stabilize replication id, #c6252d6d7f
- COUCHDB-1248: *HTTP 500* error now doesn't occurs when replicating with `?doc_ids=null`. #bea76dbf

Security

- COUCHDB-1060: Passwords are now hashed using the PBKDF2 algorithm with a configurable work factor. #7d418134

Source Repository

- The source repository was migrated from SVN to Git.

Storage System

- Fixed unnecessary conflict when deleting and creating a document in the same batch.

Test Suite

- COUCHDB-1563: Ensures urlPrefix is set in all ajax requests. #07a6af222
- COUCHDB-1389: Improved tracebacks printed by the JS CLI tests.
- COUCHDB-1339: Use shell trap to catch dying beam processes during test runs. #2921c78
- COUCHDB-1338: Start CouchDB with `port=0`. While CouchDB might be already running on the default port 5984, port number 0 let the TCP stack figure out a free port to run. #127cbe3
- COUCHDB-1321: Moved the JS test suite to the CLI.
- Improved the reliability of a number of tests.
- Fix race condition for test running on faster hardware.

URL Rewriter & Vhosts

- COUCHDB-1026: Database name is encoded during rewriting (allowing embedded /'s, etc). #272d6415

UUID Algorithms

- COUCHDB-1373: Added the utc_id algorithm #5ab712a2

Query and View Server

- COUCHDB-1491: Clenaup view tables. #c37204b7
- COUCHDB-1483: Update handlers requires valid doc ids. #72ea7e38
- COUCHDB-1445: CouchDB tries no more to delete view file if it couldn't open it, even if the error is *emfile*.
- COUCHDB-1444: Fix missed_named_view error that occurs on existed design documents and views. #b59ac98b
- COUCHDB-1372: *_stats* builtin reduce function no longer produces error for empty view result.
- COUCHDB-410: More graceful error handling for JavaScript validate_doc_update functions.
- COUCHDB-111: Improve the errors reported by the javascript view server to provide a more friendly error report when something goes wrong. #0c619ed
- Deprecate E4X support, #cdfdda2314

Windows

- COUCHDB-1482: Use correct linker flang to build *snappy_nif.dll* on Windows. #a6eaf9f1
- Allows building cleanly on Windows without cURL, #fb670f5712

13.3 1.2.x Branch

- Upgrade Notes
- Version 1.2.1
- Version 1.2.0

13.3.1 Upgrade Notes

Warning: This version drops support for the database format that was introduced in version 0.9.0. Compact your older databases (that have not been compacted for a long time) before upgrading, or they will become inaccessible.

Security changes

The interface to the `_users` and `_replicator` databases have been changed so that non-administrator users can see less information:

- In the `_users` database:
  - User documents can now only be read by the respective users, as well as administrators. Other users cannot read these documents.
  - Views can only be defined and queried by administrator users.
  - The `_changes` feed can only be queried by administrator users.
- In the `_replicator` database:
  - Documents now have a forced `owner` field that corresponds to the authenticated user that created them.
  - Non-owner users will not see confidential information like passwords or OAuth tokens in replication documents; they can still see the other contents of those documents. Administrators can see everything.
  - Views can only be defined and queried by administrators.

Database Compression

The new optional (but enabled by default) compression of disk files requires an upgrade of the on-disk format (5 -> 6) which occurs on creation for new databases and views, and on compaction for existing files. This format is not supported in previous releases, so rollback would require replication to the previous CouchDB release or restoring from backup.

Compression can be disabled by setting `compression = none` in your `local.ini [couchdb]` section, but the on-disk format will still be upgraded.

13.3.2 Version 1.2.1

Security

- Fixed CVE-2012-5641: Apache CouchDB Information disclosure via unescaped backslashes in URLs on Windows
- Fixed CVE-2012-5649: Apache CouchDB JSONP arbitrary code execution with Adobe Flash
- Fixed CVE-2012-5650: Apache CouchDB DOM based Cross-Site Scripting via Futon UI

HTTP Interface

- No longer rewrites the X-CouchDB-Requested-Path during recursive calls to the rewriter.
- Limit recursion depth in the URL rewriter. Defaults to a maximum of 100 invocations but is configurable.

Build System

- Fix couchdb start script.
- Win: fix linker invocations.

Futon

- Disable buttons that aren't available for the logged-in user.

Replication

- Fix potential timeouts.

View System

- Change use of signals to avoid broken view groups.

13.3.3 Version 1.2.0

Authentication

- Fix use of OAuth with VHosts and URL rewriting.
- OAuth secrets can now be stored in the users system database as an alternative to key value pairs in the .ini configuration. By default this is disabled (secrets are stored in the .ini) but can be enabled via the .ini configuration key *use_users_db* in the *couch_httpd_oauth* section.
- Documents in the _users database are no longer publicly readable.
- Confidential information in the _replication database is no longer publicly readable.
- Password hashes are now calculated by CouchDB. Clients are no longer required to do this manually.
- Cookies used for authentication can be made persistent by enabling the .ini configuration key *allow_persistent_cookies* in the *couch_httpd_auth* section.

Build System

- cURL is no longer required to build CouchDB as it is only used by the command line JS test runner. If cURL is available when building CouchJS you can enable the HTTP bindings by passing -H on the command line.
- Temporarily made *make check* pass with R15B. A more thorough fix is in the works (COUCHDB-1424).
- Fixed –with-js-include and –with-js-lib options.
- Added –with-js-lib-name option.

Futon

- The *Status* screen (active tasks) now displays two new task status fields: *Started on* and *Updated on*.
- Futon remembers view code every time it is saved, allowing to save an edit that amounts to a revert.

HTTP Interface

- Added a native JSON parser.
- The _active_tasks API now offers more granular fields. Each task type is now able to expose different properties.
- Added built-in changes feed filter *_view*.
- Fixes to the *_changes* feed heartbeat option which caused heartbeats to be missed when used with a filter. This caused timeouts of continuous pull replications with a filter.
- Properly restart the SSL socket on configuration changes.

Replicator

- A new replicator implementation. It offers more performance and configuration options.
- Passing non-string values to query_params is now a 400 bad request. This is to reduce the surprise that all parameters are converted to strings internally.
- Added optional field *since_seq* to replication objects/documents. It allows to bootstrap a replication from a specific source sequence number.
- Simpler replication cancellation. In addition to the current method, replications can now be canceled by specifying the replication ID instead of the original replication object/document.

Storage System

- Added optional database and view index file compression (using Google's snappy or zlib's deflate). This feature is enabled by default, but it can be disabled by adapting local.ini accordingly. The on-disk format is upgraded on compaction and new DB/view creation to support this.
- Several performance improvements, most notably regarding database writes and view indexing.
- Computation of the size of the latest MVCC snapshot data and all its supporting metadata, both for database and view index files. This information is exposed as the *data_size* attribute in the database and view group information URIs.
- The size of the buffers used for database and view compaction is now configurable.
- Added support for automatic database and view compaction. This feature is disabled by default, but it can be enabled via the .ini configuration.
- Performance improvements for the built-in changes feed filters *_doc_ids* and *_design*.

View Server

- Add CoffeeScript (http://coffeescript.org/) as a first class view server language.
- Fixed old index file descriptor leaks after a view cleanup.
- The requested_path property keeps the pre-rewrite path even when no VHost configuration is matched.
- Fixed incorrect reduce query results when using pagination parameters.
- Made icu_driver work with Erlang R15B and later.

OAuth

- Updated bundled erlang_oauth library to the latest version.

13.4 1.1.x Branch

- Version 1.1.2
- Version 1.1.1
- Version 1.1.0

13.4.1 Version 1.1.2

Security

- Fixed CVE-2012-5641: Apache CouchDB Information disclosure via unescaped backslashes in URLs on Windows.
- Fixed CVE-2012-5649: Apache CouchDB JSONP arbitrary code execution with Adobe Flash.
- Fixed CVE-2012-5650: Apache CouchDB DOM based Cross-Site Scripting via Futon UI.

HTTP Interface

- ETag of attachment changes only when the attachment changes, not the document.
- Fix retrieval of headers larger than 4k.
- Allow OPTIONS HTTP method for list requests.
- Don't attempt to encode invalid json.

Replicator

- Fix pull replication of documents with many revisions.
- Fix replication from an HTTP source to an HTTP target.

View Server

- Avoid invalidating view indexes when running out of file descriptors.

Log System

- Improvements to log messages for file-related errors.

Build System

- Don't *ln* the *couchjs* install target on Windows
- Remove ICU version dependency on Windows.
- Improve SpiderMonkey version detection.

13.4.2 Version 1.1.1

- Support SpiderMonkey 1.8.5
- Add configurable maximum to the number of bytes returned by _log.
- Allow CommonJS modules to be an empty string.
- Bump minimum Erlang version to R13B02.
- Do not run deleted validate_doc_update functions.
- ETags for views include current sequence if include_docs=true.
- Fix bug where duplicates can appear in _changes feed.
- Fix bug where update handlers break after conflict resolution.
- Fix bug with _replicator where include "filter" could crash couch.
- Fix crashes when compacting large views.
- Fix file descriptor leak in _log
- Fix missing revisions in _changes?style=all_docs.
- Improve handling of compaction at max_dbs_open limit.
- JSONP responses now send "text/javascript" for Content-Type.
- Link to ICU 4.2 on Windows.
- Permit forward slashes in path to update functions.
- Reap couchjs processes that hit reduce_overflow error.
- Status code can be specified in update handlers.
- Support provides() in show functions.
- _view_cleanup when ddoc has no views now removes all index files.
- max_replication_retry_count now supports "infinity".
- Fix replication crash when source database has a document with empty ID.
- Fix deadlock when assigning couchjs processes to serve requests.
- Fixes to the document multipart PUT API.
- Fixes regarding file descriptor leaks for databases with views.

13.4.3 Version 1.1.0

Note: All CHANGES for 1.0.2 and 1.0.3 also apply to 1.1.0.

Externals

- Added OS Process module to manage daemons outside of CouchDB.
- Added HTTP Proxy handler for more scalable externals.

Futon

- Added a "change password"-feature to Futon.

HTTP Interface

- Native SSL support.
- Added support for HTTP range requests for attachments.
- Added built-in filters for *_changes*: *_doc_ids* and *_design*.
- Added configuration option for TCP_NODELAY aka "Nagle".
- Allow POSTing arguments to *_changes*.
- Allow *keys* parameter for GET requests to views.
- Allow wildcards in vhosts definitions.
- More granular ETag support for views.
- More flexible URL rewriter.
- Added support for recognizing "Q values" and media parameters in HTTP Accept headers.
- Validate doc ids that come from a PUT to a URL.

Replicator

- Added *_replicator* database to manage replications.
- Fixed issues when an endpoint is a remote database accessible via SSL.
- Added support for continuous by-doc-IDs replication.
- Fix issue where revision info was omitted when replicating attachments.
- Integrity of attachment replication is now verified by MD5.

Storage System

- Multiple micro-optimizations when reading data.

URL Rewriter & Vhosts

- Fix for variable substituion

View Server

- Added CommonJS support to map functions.
- Added *stale=update_after* query option that triggers a view update after returning a *stale=ok* response.
- Warn about empty result caused by *startkey* and *endkey* limiting.
- Built-in reduce function *_sum* now accepts lists of integers as input.
- Added view query aliases start_key, end_key, start_key_doc_id and end_key_doc_id.

13.5 1.0.x Branch

- Version 1.0.4
- Version 1.0.3
- Version 1.0.2
- Version 1.0.1
- Version 1.0.0

13.5.1 Version 1.0.4

Security

- Fixed CVE-2012-5641: Apache CouchDB Information disclosure via unescaped backslashes in URLs on Windows.
- Fixed CVE-2012-5649: Apache CouchDB JSONP arbitrary code execution with Adobe Flash.
- Fixed CVE-2012-5650: Apache CouchDB DOM based Cross-Site Scripting via Futon UI.

Log System

- Fix file descriptor leak in *_log*.

HTTP Interface

- Fix missing revisions in *_changes?style=all_docs*.
- Fix validation of attachment names.

View System

- Avoid invalidating view indexes when running out of file descriptors.

Replicator

- Fix a race condition where replications can go stale.

13.5.2 Version 1.0.3

General

- Fixed compatibility issues with Erlang R14B02.

Etap Test Suite

- Etap tests no longer require use of port 5984. They now use a randomly selected port so they won't clash with a running CouchDB.

Futon

- Made compatible with jQuery 1.5.x.

HTTP Interface

- Fix bug that allows invalid UTF-8 after valid escapes.
- The query parameter *include_docs* now honors the parameter *conflicts*. This applies to queries against map views, _all_docs and _changes.
- Added support for inclusive_end with reduce views.

Replicator

- Enabled replication over IPv6.
- Fixed for crashes in continuous and filtered changes feeds.
- Fixed error when restarting replications in OTP R14B02.
- Upgrade ibrowse to version 2.2.0.
- Fixed bug when using a filter and a limit of 1.

Security

- Fixed OAuth signature computation in OTP R14B02.
- Handle passwords with : in them.

Storage System

- More performant queries against _changes and _all_docs when using the *include_docs* parameter.

Windows

- Windows builds now require ICU >= 4.4.0 and Erlang >= R14B03. See COUCHDB-1152, and COUCHDB-963 + OTP-9139 for more information.

13.5.3 Version 1.0.2

Security

- Fixed CVE-2010-3854: Apache CouchDB Cross Site Scripting Issue.

Futon

- Make test suite work with Safari and Chrome.
- Fixed animated progress spinner.
- Fix raw view document link due to overzealous URI encoding.
- Spell javascript correctly in loadScript(uri).

HTTP Interface

- Allow reduce=false parameter in map-only views.
- Fix parsing of Accept headers.
- Fix for multipart GET APIs when an attachment was created during a local-local replication. See COUCHDB-1022 for details.

Log System

- Reduce lengthy stack traces.
- Allow logging of native <xml> types.

Replicator

- Updated ibrowse library to 2.1.2 fixing numerous replication issues.
- Make sure that the replicator respects HTTP settings defined in the config.
- Fix error when the ibrowse connection closes unexpectedly.
- Fix authenticated replication (with HTTP basic auth) of design documents with attachments.
- Various fixes to make replication more resilient for edge-cases.

Storage System

- Fix leaking file handles after compacting databases and views.
- Fix databases forgetting their validation function after compaction.
- Fix occasional timeout errors after successfully compacting large databases.
- Fix ocassional error when writing to a database that has just been compacted.
- Fix occasional timeout errors on systems with slow or heavily loaded IO.
- Fix for OOME when compactions include documents with many conflicts.
- Fix for missing attachment compression when MIME types included parameters.
- Preserve purge metadata during compaction to avoid spurious view rebuilds.
- Fix spurious conflicts introduced when uploading an attachment after a doc has been in a conflict. See COUCHDB-902 for details.
- Fix for frequently edited documents in multi-master deployments being duplicated in _changes and _all_docs. See COUCHDB-968 for details on how to repair.
- Significantly higher read and write throughput against database and view index files.

View Server

- Don't trigger view updates when requesting *_design/doc/_info*.
- Fix for circular references in CommonJS requires.
- Made isArray() function available to functions executed in the query server.
- Documents are now sealed before being passed to map functions.
- Force view compaction failure when duplicated document data exists. When this error is seen in the logs users should rebuild their views from scratch to fix the issue. See COUCHDB-999 for details.

13.5.4 Version 1.0.1

Security

- Fixed CVE-2010-2234: Apache CouchDB Cross Site Request Forgery Attack.

Authentication

- **Enable basic-auth popup when required to access the server, to prevent** people from getting locked out.

Build and System Integration

- Included additional source files for distribution.

Futon

- User interface element for querying stale (cached) views.

HTTP Interface

- Expose *committed_update_seq* for monitoring purposes.
- Show fields saved along with _deleted=true. Allows for auditing of deletes.
- More robust Accept-header detection.

Replicator

- Added support for replication via an HTTP/HTTPS proxy.
- Fix pull replication of attachments from 0.11 to 1.0.x.
- Make the _changes feed work with non-integer seqnums.

Storage System

- Fix data corruption bug COUCHDB-844. Please see http://couchdb.apache.org/notice/1.0.1.html for details.

13.5.5 Version 1.0.0

Security

- Added authentication caching, to avoid repeated opening and closing of the users database for each request requiring authentication.

Storage System

- Small optimization for reordering result lists.
- More efficient header commits.
- Use O_APPEND to save lseeks.
- Faster implementation of pread_iolist(). Further improves performance on concurrent reads.

View Server

- Faster default view collation.
- Added option to include update_seq in view responses.

13.6 0.11.x Branch

- Version 0.11.2
- Version 0.11.1
- Version 0.11.0

13.6.1 Version 0.11.2

Security

- Fixed CVE-2010-2234: Apache CouchDB Cross Site Request Forgery Attack.
- Avoid potential DOS attack by guarding all creation of atoms.

Authentication

- User documents can now be deleted by admins or the user.

Futon

- Add some Futon files that were missing from the Makefile.

HTTP Interface

- Better error messages on invalid URL requests.

Replicator

- **Fix bug when pushing design docs by non-admins, which was hanging the** replicator for no good reason.
- **Fix bug when pulling design documents from a source that requires** basic-auth.

13.6.2 Version 0.11.1

Build and System Integration

- Output of *couchdb –help* has been improved.
- Fixed compatibility with the Erlang R14 series.
- Fixed warnings on Linux builds.
- Fixed build error when aclocal needs to be called during the build.
- Require ICU 4.3.1.
- Fixed compatibility with Solaris.

Configuration System

- Fixed timeout with large .ini files.

Futon

- Use "expando links" for over-long document values in Futon.
- Added continuous replication option.
- Added option to replicating test results anonymously to a community CouchDB instance.
- Allow creation and deletion of config entries.
- Fixed display issues with doc ids that have escaped characters.
- Fixed various UI issues.

HTTP Interface

- Mask passwords in active tasks and logging.
- Update mochijson2 to allow output of BigNums not in float form.
- Added support for X-HTTP-METHOD-OVERRIDE.
- Better error message for database names.
- Disable jsonp by default.
- Accept gzip encoded standalone attachments.
- Made max_concurrent_connections configurable.
- Made changes API more robust.
- Send newly generated document rev to callers of an update function.

JavaScript Clients

- Added tests for couch.js and jquery.couch.js
- Added changes handler to jquery.couch.js.
- Added cache busting to jquery.couch.js if the user agent is msie.
- Added support for multi-document-fetch (via _all_docs) to jquery.couch.js.
- Added attachment versioning to jquery.couch.js.
- Added option to control ensure_full_commit to jquery.couch.js.
- Added list functionality to jquery.couch.js.
- Fixed issues where bulkSave() wasn't sending a POST body.

Log System

- Log HEAD requests as HEAD, not GET.
- Keep massive JSON blobs out of the error log.
- Fixed a timeout issue.

Replication System

- Refactored various internal APIs related to attachment streaming.
- Fixed hanging replication.
- Fixed keepalive issue.

Security

- Added authentication redirect URL to log in clients.
- Fixed query parameter encoding issue in oauth.js.
- Made authentication timeout configurable.
- Temporary views are now admin-only resources.

Storage System

- Don't require a revpos for attachment stubs.
- Added checking to ensure when a revpos is sent with an attachment stub, it's correct.
- Make file deletions async to avoid pauses during compaction and db deletion.
- Fixed for wrong offset when writing headers and converting them to blocks, only triggered when header is larger than 4k.
- Preserve _revs_limit and instance_start_time after compaction.

Test Suite

- Made the test suite overall more reliable.

View Server

- Provide a UUID to update functions (and all other functions) that they can use to create new docs.
- Upgrade CommonJS modules support to 1.1.1.
- Fixed erlang filter funs and normalize filter fun API.
- Fixed hang in view shutdown.

URL Rewriter & Vhosts

- Allow more complex keys in rewriter.
- Allow global rewrites so system defaults are available in vhosts.
- Allow isolation of databases with vhosts.
- Fix issue with passing variables to query parameters.

13.6.3 Version 0.11.0

Build and System Integration

- Updated and improved source documentation.
- Fixed distribution preparation for building on Mac OS X.
- Added support for building a Windows installer as part of 'make dist'.
- Bug fix for building couch.app's module list.
- ETap tests are now run during make distcheck. This included a number of updates to the build system to properly support VPATH builds.
- Gavin McDonald setup a build-bot instance. More info can be found at http://ci.apache.org/buildbot.html

Futon

- Added a button for view compaction.
- JSON strings are now displayed as-is in the document view, without the escaping of new-lines and quotes. That dramatically improves readability of multi-line strings.
- Same goes for editing of JSON string values. When a change to a field value is submitted, and the value is not valid JSON it is assumed to be a string. This improves editing of multi-line strings a lot.
- Hitting tab in textareas no longer moves focus to the next form field, but simply inserts a tab character at the current caret position.
- Fixed some font declarations.

HTTP Interface

- Provide Content-MD5 header support for attachments.
- Added URL Rewriter handler.
- Added virtual host handling.

Replication

- Added option to implicitly create replication target databases.
- Avoid leaking file descriptors on automatic replication restarts.
- Added option to replicate a list of documents by id.
- Allow continuous replication to be cancelled.

Runtime Statistics

- Statistics are now calculated for a moving window instead of non-overlapping timeframes.
- Fixed a problem with statistics timers and system sleep.
- Moved statistic names to a term file in the priv directory.

Security

- Fixed CVE-2010-0009: Apache CouchDB Timing Attack Vulnerability.
- Added default cookie-authentication and users database.
- Added Futon user interface for user signup and login.
- Added per-database reader access control lists.
- Added per-database security object for configuration data in validation functions.
- Added proxy authentication handler

Storage System

- Adds batching of multiple updating requests, to improve throughput with many writers. Removed the now redundant couch_batch_save module.
- Adds configurable compression of attachments.

View Server

- Added optional 'raw' binary collation for faster view builds where Unicode collation is not important.
- Improved view index build time by reducing ICU collation callouts.
- Improved view information objects.
- Bug fix for partial updates during view builds.
- Move query server to a design-doc based protocol.
- Use json2.js for JSON serialization for compatiblity with native JSON.
- Major refactoring of couchjs to lay the groundwork for disabling cURL support. The new HTTP interaction acts like a synchronous XHR. Example usage of the new system is in the JavaScript CLI test runner.

13.7 0.10.x Branch

- Version 0.10.1
- Version 0.10.0

13.7.1 Version 0.10.1

Build and System Integration

- Test suite now works with the distcheck target.

Replicator

- Stability enhancements regarding redirects, timeouts, OAuth.

Query Server

- Avoid process leaks
- Allow list and view to span languages

Stats

- Eliminate new process flood on system wake

13.7.2 Version 0.10.0

Build and System Integration

- Changed *couchdb* script configuration options.
- Added default.d and local.d configuration directories to load sequence.

HTTP Interface

- Added optional cookie-based authentication handler.
- Added optional two-legged OAuth authentication handler.

Storage Format

- Add move headers with checksums to the end of database files for extra robust storage and faster storage.

View Server

- Added native Erlang views for high-performance applications.

13.8 0.9.x Branch

- Version 0.9.2
- Version 0.9.1
- Version 0.9.0

13.8.1 Version 0.9.2

Build and System Integration

- Remove branch callbacks to allow building couchjs against newer versions of Spidermonkey.

Replication

- Fix replication with 0.10 servers initiated by an 0.9 server (COUCHDB-559).

13.8.2 Version 0.9.1

Build and System Integration

- PID file directory is now created by the SysV/BSD daemon scripts.
- Fixed the environment variables shown by the configure script.
- Fixed the build instructions shown by the configure script.
- Updated ownership and permission advice in *README* for better security.

Configuration and stats system

- Corrected missing configuration file error message.
- Fixed incorrect recording of request time.

Database Core

- Document validation for underscore prefixed variables.
- Made attachment storage less sparse.
- Fixed problems when a database with delayed commits pending is considered idle, and subject to losing changes when shutdown. (COUCHDB-334)

External Handlers

- Fix POST requests.

Futon

- Redirect when loading a deleted view URI from the cookie.

HTTP Interface

- Attachment requests respect the "rev" query-string parameter.

JavaScript View Server

- Useful JavaScript Error messages.

Replication

- Added support for Unicode characters transmitted as UTF-16 surrogate pairs.
- URL-encode attachment names when necessary.
- Pull specific revisions of an attachment, instead of just the latest one.
- Work around a rare chunk-merging problem in ibrowse.
- Work with documents containing Unicode characters outside the Basic Multilingual Plane.

13.8.3 Version 0.9.0

Build and System Integration

- The *couchdb* script now supports system chainable configuration files.
- The Mac OS X daemon script now redirects STDOUT and STDERR like SysV/BSD.
- The build and system integration have been improved for portability.
- Added COUCHDB_OPTIONS to etc/default/couchdb file.
- Remove COUCHDB_INI_FILE and COUCHDB_PID_FILE from etc/default/couchdb file.
- Updated *configure.ac* to manually link *libm* for portability.
- Updated *configure.ac* to extended default library paths.
- Removed inets configuration files.
- Added command line test runner.
- Created dev target for make.

Configuration and stats system

- Separate default and local configuration files.
- HTTP interface for configuration changes.
- Statistics framework with HTTP query API.

Database Core

- Faster B-tree implementation.
- Changed internal JSON term format.
- Improvements to Erlang VM interactions under heavy load.
- User context and administrator role.
- Update validations with design document validation functions.
- Document purge functionality.
- Ref-counting for database file handles.

Design Document Resource Paths

- Added httpd_design_handlers config section.
- Moved _view to httpd_design_handlers.
- Added ability to render documents as non-JSON content-types with _show and _list functions, which are also httpd_design_handlers.

Futon Utility Client

- Added pagination to the database listing page.
- Implemented attachment uploading from the document page.
- Added page that shows the current configuration, and allows modification of option values.
- Added a JSON "source view" for document display.

- JSON data in view rows is now syntax highlighted.
- Removed the use of an iframe for better integration with browser history and bookmarking.
- Full database listing in the sidebar has been replaced by a short list of recent databases.
- The view editor now allows selection of the view language if there is more than one configured.
- Added links to go to the raw view or document URI.
- Added status page to display currently running tasks in CouchDB.
- JavaScript test suite split into multiple files.
- Pagination for reduce views.

HTTP Interface

- Added client side UUIDs for idempotent document creation
- HTTP COPY for documents
- Streaming of chunked attachment PUTs to disk
- Remove negative count feature
- Add include_docs option for view queries
- Add multi-key view post for views
- Query parameter validation
- Use stale=ok to request potentially cached view index
- External query handler module for full-text or other indexers.
- Etags for attachments, views, shows and lists
- Show and list functions for rendering documents and views as developer controlled content-types.
- Attachment names may use slashes to allow uploading of nested directories (useful for static web hosting).
- Option for a view to run over design documents.
- Added newline to JSON responses. Closes bike-shed.

Replication

- Using ibrowse.
- Checkpoint replications so failures are less expensive.
- Automatically retry of failed replications.
- Stream attachments in pull-replication.

13.9 0.8.x Branch

- Version 0.8.1-incubating
- Version 0.8.0-incubating

13.9.1 Version 0.8.1-incubating

Build and System Integration

- The *couchdb* script no longer uses *awk* for configuration checks as this was causing portability problems.
- Updated *sudo* example in *README* to use the *-i* option, this fixes problems when invoking from a directory the *couchdb* user cannot access.

Database Core

- Fix for replication problems where the write queues can get backed up if the writes aren't happening fast enough to keep up with the reads. For a large replication, this can exhaust memory and crash, or slow down the machine dramatically. The fix keeps only one document in the write queue at a time.
- Fix for databases sometimes incorrectly reporting that they contain 0 documents after compaction.
- CouchDB now uses ibrowse instead of inets for its internal HTTP client implementation. This means better replication stability.

Futon

- The view selector dropdown should now work in Opera and Internet Explorer even when it includes opt-groups for design documents. (COUCHDB-81)

JavaScript View Server

- Sealing of documents has been disabled due to an incompatibility with SpiderMonkey 1.9.
- Improve error handling for undefined values emitted by map functions. (COUCHDB-83)

HTTP Interface

- Fix for chunked responses where chunks were always being split into multiple TCP packets, which caused problems with the test suite under Safari, and in some other cases.
- Fix for an invalid JSON response body being returned for some kinds of views. (COUCHDB-84)
- Fix for connections not getting closed after rejecting a chunked request. (COUCHDB-55)
- CouchDB can now be bound to IPv6 addresses.
- The HTTP *Server* header now contains the versions of CouchDB and Erlang.

13.9.2 Version 0.8.0-incubating

Build and System Integration

- CouchDB can automatically respawn following a server crash.
- Database server no longer refuses to start with a stale PID file.
- System logrotate configuration provided.
- Improved handling of ICU shared libraries.
- The *couchdb* script now automatically enables SMP support in Erlang.
- The *couchdb* and *couchjs* scripts have been improved for portability.
- The build and system integration have been improved for portability.

Database Core

- The view engine has been completely decoupled from the storage engine. Index data is now stored in separate files, and the format of the main database file has changed.
- Databases can now be compacted to reclaim space used for deleted documents and old document revisions.
- Support for incremental map/reduce views has been added.
- To support map/reduce, the structure of design documents has changed. View values are now JSON objects containing at least a *map* member, and optionally a *reduce* member.
- View servers are now identified by name (for example *javascript*) instead of by media type.
- Automatically generated document IDs are now based on proper UUID generation using the crypto module.
- The field *content-type* in the JSON representation of attachments has been renamed to *content_type* (underscore).

Futon

- When adding a field to a document, Futon now just adds a field with an autogenerated name instead of prompting for the name with a dialog. The name is automatically put into edit mode so that it can be changed immediately.
- Fields are now sorted alphabetically by name when a document is displayed.
- Futon can be used to create and update permanent views.
- The maximum number of rows to display per page on the database page can now be adjusted.
- Futon now uses the XMLHTTPRequest API asynchronously to communicate with the CouchDB HTTP server, so that most operations no longer block the browser.
- View results sorting can now be switched between ascending and descending by clicking on the *Key* column header.
- Fixed a bug where documents that contained a @ character could not be viewed. (COUCHDB-12)
- The database page now provides a *Compact* button to trigger database compaction. (COUCHDB-38)
- Fixed portential double encoding of document IDs and other URI segments in many instances. (COUCHDB-39)
- Improved display of attachments.
- The JavaScript Shell has been removed due to unresolved licensing issues.

JavaScript View Server

- SpiderMonkey is no longer included with CouchDB, but rather treated as a normal external dependency. A simple C program (*_couchjs*) is provided that links against an existing SpiderMonkey installation and uses the interpreter embedding API.
- View functions using the default JavaScript view server can now do logging using the global *log(message)* function. Log messages are directed into the CouchDB log at *INFO* level. (COUCHDB-59)
- The global *map(key, value)* function made available to view code has been renamed to *emit(key, value)*.
- Fixed handling of exceptions raised by view functions.

HTTP Interface

- CouchDB now uses MochiWeb instead of inets for the HTTP server implementation. Among other things, this means that the extra configuration files needed for inets (such as *couch_httpd.conf*) are no longer used.
- The HTTP interface now completely supports the *HEAD* method. (COUCHDB-3)
- Improved compliance of *Etag* handling with the HTTP specification. (COUCHDB-13)
- Etags are no longer included in responses to document *GET* requests that include query string parameters causing the JSON response to change without the revision or the URI having changed.
- The bulk document update API has changed slightly on both the request and the response side. In addition, bulk updates are now atomic.
- CouchDB now uses *TCP_NODELAY* to fix performance problems with persistent connections on some platforms due to nagling.
- Including a *?descending=false* query string parameter in requests to views no longer raises an error.
- Requests to unknown top-level reserved URLs (anything with a leading underscore) now return a *unknown_private_path* error instead of the confusing *illegal_database_name*.
- The Temporary view handling now expects a JSON request body, where the JSON is an object with at least a *map* member, and optional *reduce* and *language* members.
- Temporary views no longer determine the view server based on the Content-Type header of the *POST* request, but rather by looking for a *language* member in the JSON body of the request.
- The status code of responses to *DELETE* requests is now 200 to reflect that that the deletion is performed synchronously.

INDEX

E

Emit() (built-in function), 47
emit() (built-in function), 43

F

filterfun() (built-in function), 36
FoldRows() (built-in function), 47

G

GetRow() (built-in function), 47
getRow() (built-in function), 43

I

isArray() (built-in function), 44

J

JSON (global variable or constant), 44

L

listfun() (built-in function), 35
Log() (built-in function), 48
log() (built-in function), 44

M

mapfun() (built-in function), 31

P

provides() (built-in function), 44

R

redfun() (built-in function), 32
registerType() (built-in function), 44
require() (built-in function), 45

S

Send() (built-in function), 48
send() (built-in function), 45
showfun() (built-in function), 33
Start() (built-in function), 48
start() (built-in function), 45
sum() (built-in function), 45

T

toJSON() (built-in function), 46

U

updatefun() (built-in function), 36

V

validatefun() (built-in function), 38