Package 'sofa'

May 9, 2026

Title Connector to 'CouchDB'

Description Provides an interface to the 'NoSQL' database 'CouchDB' (<http://couchdb.apache.org>). Methods are provided for managing databases within 'CouchDB', including creating/deleting/updating/transferring, and managing documents within databases. One can connect with a local 'CouchDB' instance, or a remote 'CouchDB' databases such as 'Cloudant'. Documents can be inserted directly from vectors, lists, data.frames, and 'JSON'. Targeted at 'CouchDB' v2 or greater.

Version 0.4.0

License MIT + file LICENSE

URL https://github.com/ropensci/sofa (devel)
https://docs.ropensci.org/sofa (docs)

BugReports https://github.com/ropensci/sofa/issues

Encoding UTF-8

VignetteBuilder knitr

Imports crul (>= 0.4.0), jsonlite (>= 1.5), R6 (>= 2.2.2), mime

Suggests testthat, knitr, rmarkdown

RoxygenNote 7.1.0

X-schema.org-applicationCategory Databases

X-schema.org-keywords CouchDB, database, NoSQL, documents

X-schema.org-isPartOf https://ropensci.org

NeedsCompilation no

Author Scott Chamberlain [aut, cre] (ORCID: <https://orcid.org/0000-0003-1444-9135>), rOpenSci [fnd] (https://ropensci.org/)

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2020-06-26 15:10:03 UTC

Contents

`sofa-package` *R client for CouchDB.*

Description

Relax.

About sofa

sofa provides an interface to the NoSQL database CouchDB (`http://couchdb.apache.org`). Methods are provided for managing databases within CouchDB, including creating/deleting/updating/transferring, and managing documents within databases. One can connect with a local CouchDB instance, or a remote CouchDB databases such as Cloudant (`https://cloudant.com`). Documents can be inserted directly from vectors, lists, data.frames, and JSON.

Client connections

All functions take as their first parameter a client connection object, or a **cushion**. Create the object with Cushion. You can have multiple connection objects in an R session.

CouchDB versions

sofa was built assuming CouchDB version 2 or greater. Some functionality of this package will work with versions < 2, while some may not (mango queries, see `db_query()`). I don't plan to support older CouchDB versions per se.

Digits after the decimal

If you have any concern about number of digits after the decimal in your documents, make sure to look at `digits` in your R options. The default value is 7 (see options for more informnation). You can set the value you like with e.g., `options(digits = 10)`, and get what `digits` is set to with `getOption("digits")`.

Note that in `doc_create()` we convert your document to JSON with `jsonlite::toJSON()` if given as a list, which has a `digits` parameter. We pass `getOption("digits")` to the `digits` parameter in `jsonlite::toJSON()`.

Defunct functions

- attach_get

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

`active_tasks` *active tasks*

Description

active tasks

Usage

```
active_tasks(cushion, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
# Create a CouchDB connection client
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

active_tasks(x)
active_tasks(x, as = 'json')

## End(Not run)
```

---

attachments *Work with attachments*

Description

Work with attachments

Usage

```
doc_attach_create(
  cushion,
  dbname,
  docid,
  attachment,
  attname,
  as = "list",
  ...
)

doc_attach_info(cushion, dbname, docid, attname, ...)

doc_attach_get(cushion, dbname, docid, attname = NULL, type = "raw", ...)

doc_attach_delete(cushion, dbname, docid, attname, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
dbname	(character) Database name. Required.
docid	(character) Document ID. Required.
attachment	(character) A file name. Required.
attname	(character) Attachment name. Required.
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient
type	(character) one of raw (default) or text. required.

Details

Methods:

- doc_attach_create - create an attachment
- doc_attach_info - get info (headers) for an attachment
- doc_attach_get - get an attachment. this method does not attempt to read the object into R, but only gets the raw bytes or plain text. See examples for how to read some attachment types
- doc_attach_delete - delete and attachment

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

if ("foodb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="foodb"))
}
db_create(x, dbname='foodb')

# create an attachment on an existing document
## create a document first
doc <- '{"name":"stuff", "drink":"soda"}'
doc_create(x, dbname="foodb", doc=doc, docid="asoda")

## create a csv attachment
row.names(mtcars) <- NULL
file <- tempfile(fileext = ".csv")
write.csv(mtcars, file = file, row.names = FALSE)
doc_attach_create(x, dbname="foodb", docid="asoda",
  attachment=file, attname="mtcarstable.csv")
```

```
## create a binary (png) attachment
file <- tempfile(fileext = ".png")
png(file)
plot(1:10)
dev.off()
doc_attach_create(x, dbname="foodb", docid="asoda",
  attachment=file, attname="img.png")

## create a binary (pdf) attachment
file <- tempfile(fileext = ".pdf")
pdf(file)
plot(1:10)
dev.off()
doc_attach_create(x, dbname="foodb", docid="asoda",
  attachment=file, attname="plot.pdf")

# get info for an attachment (HEAD request)
doc_attach_info(x, "foodb", docid="asoda", attname="mtcarstable.csv")
doc_attach_info(x, "foodb", docid="asoda", attname="img.png")
doc_attach_info(x, "foodb", docid="asoda", attname="plot.pdf")

# get an attachment (GET request)
res <- doc_attach_get(x, "foodb", docid="asoda",
  attname="mtcarstable.csv", type = "text")
read.csv(text = res)
doc_attach_get(x, "foodb", docid="asoda", attname="img.png")
doc_attach_get(x, "foodb", docid="asoda", attname="plot.pdf")
## OR, don't specify an attachment and list the attachments
(attchms <- doc_attach_get(x, "foodb", docid="asoda", type="text"))
jsonlite::fromJSON(attchms)

# delete an attachment
doc_attach_delete(x, "foodb", docid="asoda", attname="mtcarstable.csv")
doc_attach_delete(x, "foodb", docid="asoda", attname="img.png")
doc_attach_delete(x, "foodb", docid="asoda", attname="plot.pdf")

## End(Not run)
```

`Cushion` *sofa connection client*

Description

sofa connection client

sofa connection client

Value

An object of class `Cushion`, with variables accessible for host, port, path, transport, user, pwd, and headers. Functions are callable to get headers, and to make the base url sent with all requests.

CouchDB versions

sofa was built assuming CouchDB version 2 or greater. Some functionality of this package will work with versions < 2, while some may not (mango queries, see `db_query()`). I don't plan to support older CouchDB versions per se.

Public fields

`host` (character) host

`port` (integer) port

`path` (character) url path, if any

`transport` (character) transport schema, (httplhttps)

`user` (character) username

`pwd` (character) password

`headers` (list) named list of headers

Methods

Public methods:

- `Cushion$new()`
- `Cushion$print()`
- `Cushion$ping()`
- `Cushion$make_url()`
- `Cushion$get_headers()`
- `Cushion$get_auth()`
- `Cushion$version()`
- `Cushion$clone()`

Method `new()`**:** Create a new `Cushion` object

Usage:

```
Cushion$new(host, port, path, transport, user, pwd, headers)
```

Arguments:

`host` (character) A base URL (without the transport), e.g., `localhost`, `127.0.0.1`, or `foobar.cloudant.com`

`port` (numeric) Port. Remember that if you don't want a port set, set this parameter to `NULL`. Default: `5984`

`path` (character) context path that is appended to the end of the url. e.g., `bar` in `http://foo.com/bar`. Default: `NULL`, ignored

`transport` (character) http or https. Default: http

`user`, `pwd` (character) user name, and password. these are used in all requests. if absent, they are not passed to requests

`headers` A named list of headers. These headers are used in all requests. To use headers in individual requests and not others, pass in headers via `...` in a function call.

Returns: A new `Cushion` object

Method `print()`**:** print method for `Cushion`

Usage:

```
Cushion$print()
```

Arguments:

`x` self

`...` ignored

Method `ping()`**:** Ping the CouchDB server

Usage:

```
Cushion$ping(as = "list", ...)
```

Arguments:

`as` (character) One of list (default) or json

`...` curl options passed to crul::verb-GET

Method `make_url()`**:** Construct full base URL from the pieces in the connection object

Usage:

```
Cushion$make_url()
```

Method `get_headers()`**:** Get list of headers that will be sent with each request

Usage:

```
Cushion$get_headers()
```

Method `get_auth()`**:** Get list of auth values, user and pwd

Usage:

```
Cushion$get_auth()
```

Method `version()`**:** Get the CouchDB version as a numeric

Usage:

```
Cushion$version()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Cushion$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
# Create a CouchDB connection client
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

## metadata
x$host
x$path
```

```
x$port
x$type

## ping the CouchDB server
x$ping()

## get CouchDB version
x$version()

# create database
if (!"stuff" %in% db_list(x)) {
  db_create(x, "stuff")
}

# add documents to a database
if (!"sofadb" %in% db_list(x)) {
  db_create(x, "sofadb")
}
doc1 <- '{"name": "drink", "beer": "IPA", "score": 5}'
doc_create(x, dbname="sofadb", docid="abeer", doc1)

# bulk create
if (!"mymtcars" %in% db_list(x)) {
  db_create(x, "mymtcars")
}
db_bulk_create(x, dbname="mymtcars", doc = mtcars)
db_list(x)

## database info
db_info(x, "mymtcars")

## list dbs
db_list(x)

## all docs
db_alldocs(x, "mymtcars", limit = 3)

## changes
db_changes(x, "mymtcars")

# With auth
# x <- Cushion$new(user = 'sckott', pwd = 'sckott')

# Using Cloudant
# z <- Cushion$new(host = "ropensci.cloudant.com", transport = 'https', port = NULL,
#   user = 'ropensci', pwd = Sys.getenv('CLOUDANT_PWD'))
# z
# db_list(z)
# db_create(z, "stuff2")
# db_info(z, "stuff2")
# db_alldocs(z, "foobar")

## End(Not run)
```

databases *Work with databases in your CouchDB's.*

Description

Work with databases in your CouchDB's.

Details

There are the following functions for working with databases:

- db_create() - Create a database
- db_delete() - Delete a database
- db_info() - Get info for a database
- db_list() - List databases
- db_replicate() - Replicate a database from one couch to another

db_alldocs *List all docs in a given database.*

Description

List all docs in a given database.

Usage

```
db_alldocs(
  cushion,
  dbname,
  descending = NULL,
  startkey = NULL,
  endkey = NULL,
  limit = NULL,
  include_docs = FALSE,
  as = "list",
  disk = NULL,
  ...
)
```

Arguments

cushion	A Cushion object. Required.
dbname	Database name. (character)
descending	Return in descending order? (logical)
startkey	Document ID to start at. (character)
endkey	Document ID to end at. (character)
limit	Number document IDs to return. (numeric)
include_docs	(logical) If TRUE, returns docs themselves, in addition to IDs. Default: FALSE
as	(character) One of list (default) or json
disk	write to disk or not. By default, data is in the R session; if you give a file path, we'll write data to disk and you'll get back the file path. by default, we save in the R session
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

if ("leothelion" %in% db_list(x)) {
  invisible(db_delete(x, dbname="leothelion"))
}
db_create(x, dbname='leothelion')
db_bulk_create(x, mtcars, dbname="leothelion")

db_alldocs(x, dbname="leothelion")
db_alldocs(x, dbname="leothelion", as='json')
db_alldocs(x, dbname="leothelion", limit=2)
db_alldocs(x, dbname="leothelion", limit=2, include_docs=TRUE)

# curl options
res <- db_alldocs(x, dbname="leothelion", verbose = TRUE)

# write data to disk - useful when data is very large
## create omdb dataset first
file <- system.file("examples/omdb.json", package = "sofa")
strs <- readLines(file)
if ("omdb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="omdb"))
}
db_create(x, dbname='omdb')
invisible(db_bulk_create(x, "omdb", strs))
```

```
## get all docs, writing them to disk
res <- db_alldocs(x, dbname="omdb", disk = (f <- tempfile(fileext=".json")))
res
readLines(res, n = 10)

## End(Not run)
```

`db_bulk_create` *Create documents via the bulk API*

Description

Create documents via the bulk API

Usage

```
db_bulk_create(
  cushion,
  dbname,
  doc,
  docid = NULL,
  how = "rows",
  as = "list",
  ...
)
```

Arguments

`cushion`	A `Cushion` object. Required.
`dbname`	(character) Database name. Required.
`doc`	A data.frame, list, or JSON as a character string. Required.
`docid`	Document IDs, ignored for now, eventually, you can pass in a list, or vector to be the ids for each document created. Has to be the same length as the number of documents.
`how`	(character) One of rows (default) or columns. If rows, each row becomes a separate document; if columns, each column becomes a separate document.
`as`	(character) One of list (default) or json
`...`	Curl args passed on to `HttpClient`

Details

Note that row.names are dropped from data.frame inputs.

Value

Either a list or json (depending on as parameter), with each element an array of key:value pairs:

- ok - whether creation was successful
- id - the document id
- rev - the revision id

Examples

```
## Not run:
# initialize a CouchDB connection
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

# From a data.frame
if ("bulktest" %in% db_list(x)) {
  invisible(db_delete(x, dbname="bulktest"))
}
db_create(x, dbname="bulktest")
db_bulk_create(x, "bulktest", mtcars)

if ("bulktest2" %in% db_list(x)) {
  invisible(db_delete(x, dbname="bulktest2"))
}
db_create(x, dbname="bulktest2")
db_bulk_create(x, "bulktest2", iris)

# data.frame with 1 or more columns as neseted lists
mtcars$stuff <- list("hello_world")
mtcars$stuff2 <- list("hello_world","things")
if ("bulktest3" %in% db_list(x)) {
  invisible(db_delete(x, dbname="bulktest3"))
}
db_create(x, dbname="bulktest3")
db_bulk_create(x, "bulktest3", mtcars)

# From a json character string, or more likely, many json character strings
library("jsonlite")
strs <- as.character(parse_df(mtcars, "columns"))
if ("bulkfromchr" %in% db_list(x)) {
  invisible(db_delete(x, dbname="bulkfromchr"))
}
db_create(x, dbname="bulkfromchr")
db_bulk_create(x, "bulkfromchr", strs)

# From a list of lists
library("jsonlite")
lst <- parse_df(mtcars, tojson=FALSE)
if ("bulkfromchr" %in% db_list(x)) {
  invisible(db_delete(x, dbname="bulkfromchr"))
}
```

```
db_create(x, dbname="bulkfromchr")
db_bulk_create(x, "bulkfromchr", lst)

# iris dataset - by rows
if ("irisrows" %in% db_list(x)) {
  invisible(db_delete(x, dbname="irisrows"))
}
db_create(x, dbname="irisrows")
db_bulk_create(x, "irisrows", apply(iris, 1, as.list))

# iris dataset - by columns - doesn't quite work yet
# if ("iriscolumns" %in% db_list(x)) {
#   invisible(db_delete(x, dbname="iriscolumns"))
# }
# db_create(x, dbname="iriscolumns")
# db_bulk_create(x, "iriscolumns", parse_df(iris, "columns", tojson=FALSE), how="columns")

## End(Not run)
```

`db_bulk_get` *Query many documents at once*

Description

Query many documents at once

Usage

```
db_bulk_get(cushion, dbname, docid_rev, as = "list", ...)
```

Arguments

`cushion`	A `Cushion` object. Required.
`dbname`	(character) Database name. Required.
`docid_rev`	A list of named lists, each of which must have the slot `id`, and optionally `rev` for the revision of the document id
`as`	(character) One of list (default) or json
`...`	Curl args passed on to `HttpClient`

Value

Either a list or json (depending on `as` parameter)

Examples

```
## Not run:
# initialize a CouchDB connection
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

if ("bulkgettest" %in% db_list(x)) {
  invisible(db_delete(x, dbname="bulkgettest"))
}
db_create(x, dbname="bulkgettest")
db_bulk_create(x, "bulkgettest", mtcars)
res <- db_query(x, dbname = "bulkgettest", selector = list(cyl = 8))

# with ids only
ids <- vapply(res$docs, "[[", "", "_id")
ids_only <- lapply(ids[1:5], function(w) list(id = w))
db_bulk_get(x, "bulkgettest", docid_rev = ids_only)

# with ids and revs
ids_rev <- lapply(res$docs[1:3],
  function(w) list(id = w$`_id`, rev = w$`_rev`))
db_bulk_get(x, "bulkgettest", docid_rev = ids_rev)

## End(Not run)
```

---

db_bulk_update *Create documents via the bulk API*

Description

Create documents via the bulk API

Usage

```
db_bulk_update(
  cushion,
  dbname,
  doc,
  docid = NULL,
  how = "rows",
  as = "list",
  ...
)
```

Arguments

cushion	A Cushion object. Required.

dbname	(character) Database name. Required.
doc	For now, a data.frame only. Required.
docid	Document IDs, ignored for now, eventually, you can pass in a list, or vector to be the ids for each document created. Has to be the same length as the number of documents.
how	(character) One of rows (default) or columns. If rows, each row becomes a separate document; if columns, each column becomes a separate document.
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

Either a list or json (depending on as parameter), with each element an array of key:value pairs:

- ok - whether creation was successful
- id - the document id
- rev - the revision id

Examples

```
## Not run:
# initialize a CouchDB connection
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

row.names(mtcars) <- NULL

if ("bulktest" %in% db_list(x)) {
  invisible(db_delete(x, dbname="bulktest"))
}
db_create(x, dbname="bulktest")
db_bulk_create(x, mtcars, dbname="bulktest")

# modify mtcars
mtcars$letter <- sample(letters, NROW(mtcars), replace = TRUE)
db_bulk_update(x, "bulktest", mtcars)

# change again
mtcars$num <- 89
db_bulk_update(x, "bulktest", mtcars)

## End(Not run)
```

db_changes *List changes to a database.*

Description

Of course it doesn't make much sense to use certain options in _changes. For example, using feed=longpoll or continuous doesn't make much sense within R itself.

Usage

```
db_changes(
  cushion,
  dbname,
  descending = NULL,
  startkey = NULL,
  endkey = NULL,
  since = NULL,
  limit = NULL,
  include_docs = NULL,
  feed = "normal",
  heartbeat = NULL,
  filter = NULL,
  as = "list",
  ...
)
```

Arguments

cushion	A Cushion object. Required.
dbname	Database name. (character)
descending	Return in descending order? (logical)
startkey	Document ID to start at. (character)
endkey	Document ID to end at. (character)
since	Start the results from the change immediately after the given sequence number.
limit	Number document IDs to return. (numeric)
include_docs	(character) If "true", returns docs themselves, in addition to IDs
feed	Select the type of feed. One of normal, longpoll, or continuous. See description. (character)
heartbeat	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. (numeric (milliseconds))
filter	Reference a filter function from a design document to selectively get updates.
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

Either a list of json (depending on as parameter), with keys:

- results - Changes made to a database, length 0 if no changes. Each of these has:
  - changes - List of document's leafs with single field rev
  - id - Document ID
  - seq - Update sequence
- last_seq - Last change update sequence
- pending - Count of remaining items in the feed

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

if ("leoalion" %in% db_list(x)) {
  invisible(db_delete(x, dbname="leoalion"))
}
db_create(x, dbname='leoalion')

# no changes
res <- db_changes(x, dbname="leoalion")
res$results

# create a document
doc1 <- '{"name": "drink", "type": "water", "score": 5}'
doc_create(x, dbname="leoalion", doc1, docid="awater")

# now there's changes
res <- db_changes(x, dbname="leoalion")
res$results

# as JSON
db_changes(x, dbname="leoalion", as='json')

## End(Not run)
```

---

db_compact *Request compaction of the specified database*

Description

Request compaction of the specified database

Usage

```
db_compact(cushion, dbname, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
dbname	Database name. Required.
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Details

Compaction compresses the disk database file by performing the following operations:

- Writes a new, optimised, version of the database file, removing any unused sections from the new version during write. Because a new file is temporarily created for this purpose, you may require up to 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.

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 "/_active_tasks"

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))
# db_compact(x, dbname = "iris")

## End(Not run)
```

---

db_create *Create a database.*

Description

Create a database.

Usage

```
db_create(cushion, dbname, delifexists = FALSE, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
dbname	Database name
delifexists	If TRUE, delete any database of the same name before creating it. This is useful for testing. Default: FALSE
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

if ("leothetiger" %in% db_list(x)) {
  invisible(db_delete(x, dbname="leothetiger"))
}
db_create(x, dbname='leothetiger')

## see if its there now
db_list(x)

## End(Not run)
```

---

db_delete *Delete a database.*

Description

Delete a database.

Usage

```
db_delete(cushion, dbname, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
dbname	Database name
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

# local databasees
## create database first, then delete
db_create(x, dbname='newdb')
db_delete(x, dbname='newdb')

## with curl info while doing request
library('crul')
db_create(x, 'newdb')
db_delete(x, 'newdb', verbose = TRUE)

## End(Not run)
```

---

db_explain *Explain API*

Description

Explain API

Usage

```
db_explain(
  cushion,
  dbname,
  query = NULL,
  selector = NULL,
  limit = NULL,
  skip = NULL,
  sort = NULL,
  fields = NULL,
  use_index = NULL,
  as = "list",
  ...
)
```

Arguments

cushion	A Cushion object. Required.
dbname	Database name
query	(character) instead of using the other parameters, you can compose one R list or json blob here
selector	(json) - JSON object describing criteria used to select documents. More information provided in the section on selector syntax. See the query_tutorial in this package, and the selectors docs http://docs.couchdb.org/en/2.0.0/api/database/find.html#find-selectors
limit	(number) - Maximum number of results returned. Default: 25 Optional
skip	(number) - Skip the first 'n' results, where 'n' is the value specified. Optional
sort	(json) - JSON array following sort syntax. Optional. See http://docs.couchdb.org/en/2.0.0/api/database/find.html#find-sort For some reason, sort doesn't work often, not sure why.
fields	(json) - JSON array specifying which fields of each object should be returned. If it is omitted, the entire object is returned. More information provided in the section on filtering fields. Optional See http://docs.couchdb.org/en/2.0.0/api/database/find.html#find-filter
use_index	(json) - Instruct a query to use a specific index. Specified either as <design_document> or ["<design_document>", "<index_name>"]. Optional
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
## create a connection
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

file <- system.file("examples/omdb.json", package = "sofa")
strs <- readLines(file)

## create a database
if ("omdb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="omdb"))
}
db_create(x, dbname='omdb')

## add some documents
invisible(db_bulk_create(x, "omdb", strs))
```

```
## query all in one json blob
db_explain(x, dbname = "omdb", query = '{
  "selector": {
    "_id": {
      "$gt": null
    }
  }
}')

## End(Not run)
```

---

db_index *Create and get database indexes*

Description

Create and get database indexes

Usage

```
db_index(cushion, dbname, as = "list", ...)

db_index_create(cushion, dbname, body, as = "list", ...)

db_index_delete(cushion, dbname, design, index_name, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
dbname	(character) Database name, required
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient
body	(named list) index fields, required
design	(character) Design document name
index_name	(character) index name

Value

JSON as a character string or a list (determined by the as parameter)

Body parameters

- index (json) - JSON object describing the index to create.
- ddoc (string) - Name of the design document in which the index will be created. By default, each index will be created in its own design document. Indexes can be grouped into design documents for efficiency. However, a change to one index in a design document will invalidate all other indexes in the same document (similar to views). Optional

- name (string) - Name of the index. If no name is provided, a name will be generated automatically. Optional
- type (string) - Can be "json" or "text". Defaults to json. Geospatial indexes will be supported in the future. Optional Text indexes are supported via a third party library Optional
- partial_filter_selector (json) - A selector to apply to documents at indexing time, creating a partial index. Optional

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

# create a database first
if ("testing" %in% db_list(x)) {
  invisible(db_delete(x, dbname="testing"))
}
db_create(x, "testing")

# get indexes
db_index(x, "testing")

# create indexes
body <- list(index = list(fields = I("foo")), name = "foo-index", type = "json")
db_index_create(x, "testing", body = body)

# get indexes, after creating another index
db_index(x, "testing")

# delete an index
res <- db_index(x, "testing")
db_index_delete(x, "testing", res$indexes[[2]]$ddoc, res$indexes[[2]]$name)
## and it's gone
db_index(x, "testing")

## End(Not run)
```

`db_info` *List database info.*

Description

List database info.

Usage

```
db_info(cushion, dbname, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
dbname	Database name
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

if ("sofadb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="sofadb"))
}
db_create(x, dbname='sofadb')

db_info(x, dbname="sofadb")
db_info(x, dbname="sofadb", as='json')

## End(Not run)
```

---

db_list *List all databases.*

Description

List all databases.

Usage

```
db_list(cushion, simplify = TRUE, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
simplify	(logical) Simplify to character vector, ignored if as="json"
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

db_list(x)
db_list(x, as = 'json')

## End(Not run)
```

---

db_query *Query a database.*

Description

Query a database.

Usage

```
db_query(
  cushion,
  dbname,
  query = NULL,
  selector = NULL,
  limit = NULL,
  skip = NULL,
  sort = NULL,
  fields = NULL,
  use_index = NULL,
  r = NULL,
  bookmark = NULL,
  update = NULL,
  stable = NULL,
  stale = NULL,
  execution_stats = FALSE,
  as = "list",
  ...
)
```

Arguments

cushion	A Cushion object. Required.
dbname	Database name
query	(character) instead of using the other parameters, you can compose one R list or json blob here
selector	(list/json) - JSON object describing criteria used to select documents. More information provided in the section on selector syntax. See the query_tutorial in this package, and the selectors docs http://docs.couchdb.org/en/2.0.0/api/database/find.html#find-selectors
limit	(numeric) - Maximum number of results returned. Default is 25. Optional
skip	(numeric) - Skip the first 'n' results, where 'n' is the value specified. Optional
sort	(json) - JSON array following sort syntax. Optional. See http://docs.couchdb.org/en/2.0.0/api/database/find.html#find-sort For some reason, sort doesn't work often, not sure why.
fields	(json) - JSON array specifying which fields of each object should be returned. If it is omitted, the entire object is returned. More information provided in the section on filtering fields. Optional See http://docs.couchdb.org/en/2.0.0/api/database/find.html#find-filter
use_index	(json) - Instruct a query to use a specific index. Specified either as <design_document> or ["<design_document>", "<index_name>"]. Optional
r	(numeric) Read quorum needed for the result. This defaults to 1, in which case the document found in the index is returned. If set to a higher value, each document is read from at least that many replicas before it is returned in the results. This is likely to take more time than using only the document stored locally with the index. Optional, default: 1
bookmark	(character) A string that enables you to specify which page of results you require. Used for paging through result sets. Every query returns an opaque string under the bookmark key that can then be passed back in a query to get the next page of results. If any part of the selector query changes between requests, the results are undefined. Optional, default: NULL
update	(logical) Whether to update the index prior to returning the result. Default is true. Optional
stable	(logical) Whether or not the view results should be returned from a "stable" set of shards. Optional
stale	(character) Combination of update=FALSE and stable=TRUE options. Possible options: "ok", "FALSE" (default). Optional
execution_stats	(logical) Include execution statistics in the query response. Optional, default: FALSE
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
## create a connection
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

file <- system.file("examples/omdb.json", package = "sofa")
strs <- readLines(file)

## create a database
if ("omdb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="omdb"))
}
db_create(x, dbname='omdb')

## add some documents
invisible(db_bulk_create(x, "omdb", strs))

## query all in one json blob
db_query(x, dbname = "omdb", query = '{
  "selector": {
    "_id": {
      "$gt": null
    }
  }
}')

## query with each parameter
db_query(x, dbname = "omdb",
  selector = list(`_id` = list(`$gt` = NULL)))

db_query(x, dbname = "omdb",
  selector = list(`_id` = list(`$gt` = NULL)), limit = 3)

# fields
## single field works
db_query(x, dbname = "omdb",
  selector = list(`_id` = list(`$gt` = NULL)), limit = 3,
  fields = c('_id', 'Actors', 'imdbRating'))

## as well as many fields
db_query(x, dbname = "omdb",
  selector = list(`_id` = list(`$gt` = NULL)), limit = 3,
  fields = '_id')

## other queries
db_query(x, dbname = "omdb",
  selector = list(Year = list(`$gt` = "2013")))

db_query(x, dbname = "omdb", selector = list(Rated = "R"))
```

```
db_query(x, dbname = "omdb",
  selector = list(Rated = "PG", Language = "English"))

db_query(x, dbname = "omdb", selector = list(
  `$or` = list(
    list(Director = "Jon Favreau"),
    list(Director = "Spike Lee")
  )
), fields = c("_id", "Director"))

## when selector vars are of same name, use a JSON string
## b/c R doesn't let you have a list with same name slots
db_query(x, dbname = "omdb", query = '{
  "selector": {
    "Year": {"$gte": "1990"},
    "Year": {"$lte": "2000"},
    "$not": {"Year": "1998"}
  },
  "fields": ["_id", "Year"]
}')

## regex
db_query(x, dbname = "omdb", selector = list(
  Director = list(`$regex` = "^R")
), fields = c("_id", "Director"))


## End(Not run)
```

`db_replicate` *Upload (replicate) a local database to a remote database server, e.g., Cloudant, Iriscouch*

Description

Upload (replicate) a local database to a remote database server, e.g., Cloudant, Iriscouch

Usage

```
db_replicate(from, to, dbname, createdb = FALSE, as = "list", ...)
```

Arguments

`from`	Couch to replicate from. An object of class Cushion. Required.
`to`	Remote couch to replicate to. An object of class Cushion. Required.
`dbname`	(character) Database name. Required.
`createdb`	If `TRUE`, the function creates the db on the remote server before uploading. The db has to exist before uploading, so either you do it separately or this fxn can do it for you. Default: `FALSE`

as	(character) One of list (default) or json
...	Curl args passed on to crul::HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
if (interactive()) {
## create a connection
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

# Create a database locally
db_list(x)
if ("hello_earth" %in% db_list(x)) {
  invisible(db_delete(x, dbname="hello_earth"))
}
db_create(x, 'hello_earth')

## replicate to a remote server
z <- Cushion$new(host = "ropensci.cloudant.com", transport = 'https',
  port = NULL, user = 'ropensci', pwd = Sys.getenv('CLOUDANT_PWD'))

## do the replication
db_replicate(x, z, dbname = "hello_earth")

## check changes on the remote
db_list(z)
db_changes(z, dbname = "hello_earth")

## make some changes on the remote
doc_create(z, dbname = "hello_earth",
  '{"language":"python","library":"requests"}', 'stuff')
changes(z, dbname = "hello_earth")

## create another document, and try to get it
doc_create(z, dbname = "hello_earth", doc = '{"language":"R"}',
  docid="R_rules")
doc_get(z, dbname = "hello_earth", docid='R_rules')

## cleanup - delete the database
db_delete(z, 'hello_earth')
}

## End(Not run)
```

db_revisions *Get document revisions.*

Description

Get document revisions.

Usage

```
db_revisions(cushion, dbname, docid, simplify = TRUE, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
dbname	Database name
docid	Document ID
simplify	(logical) Simplify to character vector of revision ids. If FALSE, gives back availability info too. Default: TRUE
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

if ("sofadb" %in% db_list(x)) {
 db_delete(x, dbname = "sofadb")
}
db_create(x, dbname = "sofadb")

doc1 <- '{"name": "drink", "beer": "IPA", "score": 5}'
doc_create(x, dbname="sofadb", doc1, docid="abeer")
doc_create(x, dbname="sofadb", doc1, docid="morebeer", as='json')

db_revisions(x, dbname="sofadb", docid="abeer")
db_revisions(x, dbname="sofadb", docid="abeer", simplify=FALSE)
db_revisions(x, dbname="sofadb", docid="abeer", as='json')
db_revisions(x, dbname="sofadb", docid="abeer", simplify=FALSE, as='json')

## End(Not run)
```

`design` *Work with design documents*

Description

Work with design documents

Usage

```
design_create(
  cushion,
  dbname,
  design,
  fxnname,
  key = "null",
  value = "doc",
  as = "list",
  ...
)

design_create_(cushion, dbname, design, fxnname, fxn, as = "list", ...)

design_delete(cushion, dbname, design, as = "list", ...)

design_get(cushion, dbname, design, as = "list", ...)

design_head(cushion, dbname, design, ...)

design_info(cushion, dbname, design, ...)
```

Arguments

`cushion`	A `Cushion` object. Required.
`dbname`	(character) Database name. required.
`design`	(character) Design document name. this is the design name without `_design/`, which is prepended internally. required.
`fxnname`	(character) A function name. required for `view_put` and `view_put_`
`key, value`	(character) a key and value, see Examples and Details
`as`	(character) One of list (default) or json
`...`	Curl args passed on to `HttpClient`
`fxn`	(character) a javascript function. required for `view_put_`

Details

`design_create` is a slightly easier interface to creating design documents; it just asks for a function name, the key and a value, then we create the function for you internally. TO have more flexibility use `view_put_` (with underscore on the end) to write the function yourself.

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

file <- system.file("examples/omdb.json", package = "sofa")
strs <- readLines(file)

## create a database
if ("omdb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="omdb"))
}
db_create(x, dbname='omdb')

## add the documents
invisible(db_bulk_create(x, "omdb", strs))

# Create a view, the easy way, but less flexible
design_create(x, dbname='omdb', design='view1', fxnname="foobar1")
design_create(x, dbname='omdb', design='view2', fxnname="foobar2",
  value="doc.Country")
design_create(x, dbname='omdb', design='view5', fxnname="foobar3",
  value="[doc.Country,doc.imdbRating]")

# the harder way, write your own function, but more flexible
design_create_(x, dbname='omdb', design='view22',
  fxnname = "stuffthings", fxn = "function(doc){emit(null,doc.Country)}")

# Delete a view
design_delete(x, dbname='omdb', design='view1')

# Get info on a design document
## HEAD request, returns just response headers
design_head(x, dbname='omdb', design='view2')
design_head(x, dbname='omdb', design='view5')
## GET request, returns information about the design document
design_info(x, dbname='omdb', design='view2')
design_info(x, dbname='omdb', design='view5')

# Get a design document (GET request)
design_get(x, dbname='omdb', design='view2')
design_get(x, dbname='omdb', design='view5')

# Search using a view
res <- design_search(x, dbname='omdb', design='view2', view='foobar2')
head(
  do.call(
```

```
    "rbind.data.frame",
    lapply(res$rows, function(x) Filter(length, x))
  )
)

res <- design_search(x, dbname='omdb', design='view5', view='foobar3')
head(
  structure(do.call(
    "rbind.data.frame",
    lapply(res$rows, function(x) x$value)
  ), .Names = c('Country', 'imdbRating'))
)

## End(Not run)
```

`design_search` *Search design documents*

Description

Search design documents

Usage

```
design_search(
  cushion,
  dbname,
  design,
  view,
  params = list(),
  body = list(),
  as = "list",
  ...
)

design_search_many(cushion, dbname, design, view, queries, as = "list", ...)
```

Arguments

`cushion`	A `Cushion` object. Required.
`dbname`	(character) Database name. required.
`design`	(character) Design document name. this is the design name without `_design/`, which is prepended internally. required.
`view`	(character) a view, same as `fxn` param in `design_create_()`. required.
`params`	query parameters. a named list
`body`	same as `params`, but if any given, a POST request is sent (if body non-NULL, `params` also sent). a named list

as	(character) One of list (default) or json
...	Curl args passed on to HttpClient
queries	a list of named lists of queries

Value

JSON as a character string or a list (determined by the as parameter)

Options to pass to params, body, or queries params

- conflicts (logical) Includes conflicts information in response. Ignored if include_docs isn't TRUE. Default: FALSE
- descending (logical) Return the documents in descending by key order. Default: FALSE
- endkey,end_key (list) Stop returning records when the specified key is reached. Optional. end_key is an alias for endkey
- endkey_docid,end_key_doc_id (character) Stop returning records when the specified document ID is reached. Requires endkey to be specified for this to have any effect. Optional. end_key_doc_id is an alias for endkey_docid
- group (logical) Group the results using the reduce function to a group or single row. Default: FALSE
- group_level (integer) Specify the group level to be used. Optional
- include_docs (logical) Include the associated document with each row. Default: FALSE.
- attachments (logical) Include the Base64-encoded content of attachments in the documents that are included if include_docs is TRUE. Ignored if include_docs isn't TRUE. Default: FALSE
- att_encoding_info (logical) Include encoding information in attachment stubs if include_docs is TRUE and the particular attachment is compressed. Ignored if include_docs isn't TRUE. Default: FALSE.
- inclusive_end (logical) Specifies whether the specified end key should be included in the result. Default: TRUE
- key (list) Return only documents that match the specified key. Optional
- keys (list) Return only documents where the key matches one of the keys specified in the array. Optional
- limit (integer) Limit the number of the returned documents to the specified number. Optional
- reduce (logical) Use the reduction function. Default: TRUE
- skip (integer) Skip this number of records before starting to return the results. Default: 0
- sorted (logical) Sort returned rows (see Sorting Returned Rows). Setting this to FALSE offers a performance boost. The total_rows and offset fields are not available when this is set to FALSE. Default: TRUE
- stale (character) Allow the results from a stale view to be used. Supported values: ok and update_after. Optional
- startkey,start_key (list) Return records starting with the specified key. Optional. start_key is an alias for startkey

- startkey_docid,start_key_doc_id (character) Return records starting with the specified document ID. Requires startkey to be specified for this to have any effect. Optional. `start_key_doc_id` is an alias for `startkey_docid`
- update_seq (logical) Response includes an update_seq value indicating which sequence id of the database the view reflects. Default: `FALSE`

References

https://docs.couchdb.org/en/latest/api/ddoc/views.html

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

file <- system.file("examples/omdb.json", package = "sofa")
strs <- readLines(file)

## create a database
if ("omdb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="omdb"))
}
db_create(x, dbname='omdb')

## add the documents
invisible(db_bulk_create(x, "omdb", strs))

# Create a view, the easy way, but less flexible
design_create(x, dbname='omdb', design='view1', fxnname="foobar1")
design_create(x, dbname='omdb', design='view2', fxnname="foobar2",
  value="doc.Country")
design_create(x, dbname='omdb', design='view5', fxnname="foobar3",
  value="[doc.Country,doc.imdbRating]")
design_create_(x, dbname='omdb', design='view6', fxnname="foobar4",
  fxn = "function(doc){emit(doc._id,doc.Country)}")

# Search using a view
compact <- function(l) Filter(Negate(is.null), l)
res <- design_search(x, dbname='omdb', design='view2', view ='foobar2')
head(
  do.call(
    "rbind.data.frame",
    Filter(
      function(z) length(z) == 2,
      lapply(res$rows, function(x) compact(x[names(x) %in% c('id', 'value')]))
    )
  )
)

res <- design_search(x, dbname='omdb', design='view5', view = 'foobar3')
```

```
head(
  structure(do.call(
    "rbind.data.frame",
    lapply(res$rows, function(x) x$value)
  ), .Names = c('Country', 'imdbRating'))
)

# query parameters
## limit
design_search(x, dbname='omdb', design='view5', view = 'foobar3',
  params = list(limit = 5))
## limit and skip
design_search(x, dbname='omdb', design='view5', view = 'foobar3',
  params = list(limit = 5, skip = 3))
## with start and end keys
### important: the key strings have to be in JSON, so here e.g.,
###  need to add escaped double quotes
res <- design_search(
  cushion = x,
  dbname = 'omdb',
  design = 'view6',
  view = 'foobar4',
  params = list(
    startkey = "\"c25bbf4fef99408b3e1115374a03f642\"",
    endkey = "\"c25bbf4fef99408b3e1115374a040f11\""
  )
)

# POST request
ids <- vapply(db_alldocs(x, dbname='omdb')$rows[1:3], "[[", "", "id")
res <- design_search(x, dbname='omdb', design='view6', view = 'foobar4',
  body = list(keys = ids), verbose = TRUE)
res

# Many queries at once in a POST request
queries <- list(
  list(keys = ids),
  list(limit = 3, skip = 2)
)
design_search_many(x, 'omdb', 'view6', 'foobar4', queries)

## End(Not run)
```

---

documents *Work with documents in your CouchDB's.*

Description

Work with documents in your CouchDB's.

Details

If you are writing a complicated javascript function, better to do that in the Futon CouchDB interface or otherwise.

There are the following functions for working with documents:

- `doc_create` - Create a document, with or without an ID
- `doc_update` - Update a document
- `doc_get` - Get a document
- `doc_delete` - Delete a document
- `doc_head` - Get headers for a document
- `doc_attach_create` - Attach something to a document
- `doc_attach_info` - Get info on an attachment
- `doc_attach_get` - Fetch an attachment
- `doc_attach_delete` - Delete an attachment
- `db_alldocs` - Get all documents
- `db_revisions` - Get revisions for a document

`doc_create` *Create documents to a database.*

Description

Create documents to a database.

Usage

```
doc_create(cushion, dbname, doc, docid = NULL, how = "rows", as = "list", ...)
```

Arguments

`cushion`	A `Cushion` object. Required.
`dbname`	Database name3
`doc`	Document content, can be character string or a list. The character type can be XML as well, if embedded in JSON. When the document is retrieved via `doc_get()`, the XML is given back and you can parse it as normal.
`docid`	Document ID
`how`	(character) One of rows (default) or columns. If rows, each row becomes a separate document; if columns, each column becomes a separate document.
`as`	(character) One of list (default) or json
`...`	Curl args passed on to `HttpClient`

Details

Documents can have attachments just like email. There are two ways to use attachments: the first one is via a separate REST call (see doc_attach_create()); the second is inline within your document, you can do so with this fxn. See https://docs.couchdb.org/en/latest/api/document/attachments.html for help on formatting json appropriately.

Note that you can create documents from a data.frame with this function, where each row or column is a separate document. However, this function does not use the bulk API https://couchdb.readthedocs.org/en/latest/api/database/bulk-api.html#db-bulk-docs

- see db_bulk_create() and db_bulk_update() to create or update documents with the bulk API - which should be much faster for a large number of documents.

Value

JSON as a character string or a list (determined by the as parameter)

Digits after the decimal

If you have any concern about number of digits after the decimal in your documents, make sure to look at digits in your R options. The default value is 7 (see options for more informnation). You can set the value you like with e.g., options(digits = 10), and get what digits is set to with getOption("digits").

Note that in doc_create() we convert your document to JSON with jsonlite::toJSON() if given as a list, which has a digits parameter. We pass getOption("digits") to the digits parameter in jsonlite::toJSON()

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

if ("sofadb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="sofadb"))
}
db_create(x, 'sofadb')

# write a document WITH a name (uses PUT)
doc1 <- '{"name": "drink", "beer": "IPA", "score": 5}'
doc_create(x, dbname="sofadb", doc1, docid="abeer")
doc_create(x, dbname="sofadb", doc1, docid="morebeer", as='json')
doc_get(x, dbname = "sofadb", docid = "abeer")
## with factor class values
doc2 <- list(name = as.factor("drink"), beer = "stout", score = 4)
doc_create(x, doc2, dbname="sofadb", docid="nextbeer", as='json')
doc_get(x, dbname = "sofadb", docid = "nextbeer")

# write a json document WITHOUT a name (uses POST)
doc2 <- '{"name": "food", "icecream": "rocky road"}'
```

```
doc_create(x, doc2, dbname="sofadb")
doc3 <- '{"planet": "mars", "size": "smallish"}'
doc_create(x, doc3, dbname="sofadb")
## assigns a UUID instead of a user given name
db_alldocs(x, dbname = "sofadb")

# write an xml document WITH a name (uses PUT). xml is written as xml in
# couchdb, just wrapped in json, when you get it out it will be as xml
doc4 <- "<top><a/><b/><c><d/><e>bob</e></c></top>"
doc_create(x, doc4, dbname="sofadb", docid="somexml")
doc_get(x, dbname = "sofadb", docid = "somexml")

# You can pass in lists that autoconvert to json internally
doc1 <- list(name = "drink", type = "soda", score = 9)
doc_create(x, dbname="sofadb", doc1, docid="gooddrink")

# Write directly from a data.frame
## Each row or column becomes a separate document
### by rows
if ("test" %in% db_list(x)) {
  invisible(db_delete(x, dbname="test"))
}
db_create(x, dbname = "test")
doc_create(x, mtcars, dbname="test", how="rows")
doc_create(x, mtcars, dbname="test", how="columns")

if ("testiris" %in% db_list(x)) {
  invisible(db_delete(x, dbname="testiris"))
}
db_create(x, dbname = "testiris")
head(iris)
doc_create(x, iris, dbname = "testiris")

## End(Not run)
```

`doc_delete` *Delete a document in a database.*

Description

Delete a document in a database.

Usage

```
doc_delete(cushion, dbname, docid, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
dbname	Database name. (character)

docid	Document ID (character)
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

# create a database
if ("sofadb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="sofadb"))
}
db_create(x, dbname='sofadb')

doc3 <- "<top><a/><b/><c><d/><e>bob</e></c></top>"
doc_create(x, dbname="sofadb", doc=doc3, docid="newnewxml")
doc_delete(x, dbname="sofadb", docid="newnewxml")

# wrong docid name
doc_create(x, dbname="sofadb", doc=doc3, docid="newxml")
# doc_delete(x, dbname="sofadb", docid="wrongname")

## End(Not run)
```

---

doc_get *Get a document from a database.*

Description

Get a document from a database.

Usage

```
doc_get(
  cushion,
  dbname,
  docid,
  rev = NULL,
  attachments = FALSE,
  deleted = FALSE,
  revs = FALSE,
```

```
  revs_info = FALSE,
  conflicts = FALSE,
  deleted_conflicts = FALSE,
  local_seq = FALSE,
  as = "list",
  ...
)
```

Arguments

cushion	A Cushion object. Required.
dbname	Database name
docid	Document ID
rev	Revision id of the document to get. If NULL, gets current revision
attachments	(logical) Whether to include _attachments field.
deleted	(logical) Whether to include _deleted field.
revs	(logical) Whether to include _revisions field.
revs_info	(logical) Whether to include _revs_info field.
conflicts	(logical) Whether to include _conflicts field.
deleted_conflicts	(logical) Whether to include _deleted_conflicts field.
local_seq	(logical) Whether to include _local_seq field.
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

if ("sofadb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="sofadb"))
}
db_create(x, dbname="sofadb")

# create a document
doc1 <- '{"name": "drink", "type": "drink", "score": 5}'
doc_create(x, dbname="sofadb", doc1, docid="asoda")

doc_get(x, dbname="sofadb", docid="asoda")
revs <- db_revisions(x, dbname="sofadb", docid="asoda")
```

```
doc_get(x, dbname="sofadb", docid="asoda", rev=revs[1])
doc_get(x, dbname="sofadb", docid="asoda", as='json')
doc_get(x, dbname="sofadb", docid="asoda", revs=TRUE)
doc_get(x, dbname="sofadb", docid="asoda", revs=TRUE, local_seq=TRUE)

## End(Not run)
```

---

doc_head *Get header info for a document*

Description

Get header info for a document

Usage

```
doc_head(cushion, dbname, docid, ...)
```

Arguments

cushion	A Cushion object. Required.
dbname	(character) Database name. Required.
docid	(character) Document ID. Required.
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

# create a database
if ("sofadb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="sofadb"))
}
db_create(x, dbname='sofadb')

# create a document
doc1 <- '{"name": "drink", "beer": "IPA", "score": 5}'
doc_create(x, dbname="sofadb", doc1, docid="abeer")

# run doc_head
doc_head(x, dbname="sofadb", docid="abeer")

## End(Not run)
```

doc_update *Update a document.*

Description

Update a document.

Usage

```
doc_update(cushion, dbname, doc, docid, rev, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
dbname	(character) Database name. Required.
doc	(character) Document content. Required.
docid	(character) Document ID. Required.
rev	(character) Revision id. Required.
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Details

Internally, this function adds in the docid and revision id, required to do a document update

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

if ("sofadb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="sofadb"))
}
db_create(x, dbname='sofadb')

doc1 <- '{"name":"drink","beer":"IPA"}'
doc_create(x, dbname="sofadb", doc=doc1, docid="b_beer")
doc_get(x, dbname = "sofadb", docid = "b_beer")
revs <- db_revisions(x, dbname = "sofadb", docid = "b_beer")
doc2 <- '{"name":"drink","beer":"IPA","note":"yummy","note2":"yay"}'
doc_update(x, dbname="sofadb", doc=doc2, docid="b_beer", rev=revs[1])
```

```
db_revisions(x, dbname = "sofadb", docid = "b_beer")

## End(Not run)
```

---

doc_upsert *Create a new document or update an existing one*

Description

Create a new document or update an existing one

Usage

```
doc_upsert(cushion, dbname, doc, docid)
```

Arguments

cushion	A Cushion object. Required.
dbname	(character) Database name. Required.
doc	(character) Document content. Required.
docid	(character) Document ID. Required.

Details

Internally, this function attempts to update a document with the given name.
If the document does not exist, it is created

Value

JSON as a character string or a list (determined by the as parameter)

Author(s)

George Kritikos

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

if ("sofadb" %in% db_list(x)) {
  invisible(db_delete(x, dbname="sofadb"))
}
db_create(x, 'sofadb')

# create a document
```

```
doc1 <- '{"name": "drink", "beer": "IPA", "score": 5}'
doc_upsert(x, dbname="sofadb", doc1, docid="abeer")

#update the document
doc2 <- '{"name": "drink", "beer": "lager", "score": 6}'
doc_upsert(x, dbname="sofadb", doc2, docid="abeer")


doc_get(x, dbname = "sofadb", docid = "abeer")

## End(Not run)
```

`membership` *membership*

Description

membership

Usage

```
membership(cushion, as = "list", ...)
```

Arguments

`cushion`	A `Cushion` object. Required.
`as`	(character) One of list (default) or json
`...`	Curl args passed on to `HttpClient`

Value

JSON as a character string or a list (determined by the `as` parameter)

Examples

```
## Not run:
# Create a CouchDB connection client
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

membership(x)
membership(x, as = 'json')

## End(Not run)
```

parse_df *Parse data.frame to json or list by row or column*

Description

Parse data.frame to json or list by row or column

Usage

```
parse_df(dat, how = "rows", tojson = TRUE, ...)
```

Arguments

dat	(data.frame) A data.frame, matrix, or tbl_df
how	(character) One of rows (default) or columns. If rows, each row becomes a separate document; if columns, each column becomes a separate document.
tojson	(logical) If TRUE (default) convert to json - if FALSE, to lists
...	Further args passed on to jsonlite::toJSON()

Details

Parse data.frame to get either rows or columns, each as a list or json string

Examples

```
## Not run:
parse_df(mtcars, how="rows")
parse_df(mtcars, how="columns")
parse_df(mtcars, how="rows", tojson=FALSE)
parse_df(mtcars, how="columns", tojson=FALSE)

## End(Not run)
```

ping *Ping a CouchDB server*

Description

Ping a CouchDB server

Usage

```
ping(cushion, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
# initialize a CouchDB connection
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

# call ping on the cushion object, or pass the cushion to ping()
x$ping()
ping(x)
ping(x, as = "json")

## End(Not run)
```

---

restart *Restart your Couchdb instance*

Description

Restart your Couchdb instance

Usage

```
restart(cushion = "localhost", as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

# restart(x)

## End(Not run)
```

`session` *session*

Description

session

Usage

```
session(cushion, as = "list", ...)
```

Arguments

`cushion`	A `Cushion` object. Required.
`as`	(character) One of list (default) or json
`...`	Curl args passed on to `HttpClient`

Value

JSON as a character string or a list (determined by the `as` parameter)

Examples

```
## Not run:
# Create a CouchDB connection client
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

session(x)
session(x, as = 'json')

## End(Not run)
```

uuids *Get uuids.*

Description

Get uuids.

Usage

```
uuids(cushion, count = 1, as = "list", ...)
```

Arguments

cushion	A Cushion object. Required.
count	(numeric) Number of uuids to return. Default: 1
as	(character) One of list (default) or json
...	Curl args passed on to HttpClient

Value

JSON as a character string or a list (determined by the as parameter)

Examples

```
## Not run:
# Create a CouchDB connection client
user <- Sys.getenv("COUCHDB_TEST_USER")
pwd <- Sys.getenv("COUCHDB_TEST_PWD")
(x <- Cushion$new(user=user, pwd=pwd))

uuids(x)
uuids(x, as = 'json')

## End(Not run)
```

Index