# python-cloudant documentation

***Release 2.11.0***

**IBM**

**Jan 21, 2019**

# Contents

This is the official Cloudant client library for Python.

CHAPTER 1

# Compatibility

This library can be used with the following databases

- IBM Cloudant® Database-as-a-Service
- IBM Cloudant® Data Layer Local Edition (Cloudant Local)
- Apache CouchDB™

Note that some features are Cloudant specific.

This library has been tested with the following versions of Python

- Python™ 2.7
- Python™ 3.5

CHAPTER 2

# Getting started

Now it's time to begin doing some work with Cloudant and Python. For working code samples of any of the API's please go to our test suite.

## 2.1 Connections

In order to manage a connection you must first initialize the connection by constructing either a `Cloudant` or `CouchDB` client. Since connecting to the Cloudant managed service provides extra end points as compared to a CouchDB server, we provide the two different client implementations in order to connect to the desired database service. Once the client is constructed, you follow that up by connecting to the server, performing your tasks, and then disconnecting from the server.

Later in the *Context managers* section we will see how to simplify this process through the use of the Python *with* statement.

Note: If you require retrying requests after an HTTP 429 error, the `Replay429Adapter` can be added when constructing a `Cloudant` client and configured with an initial back off and retry count.

Note: Currently, the connect and read timeout will wait forever for a HTTP connection or a response on all requests. A timeout can be set using the `timeout` argument when constructing a client.

### 2.1.1 Connecting with a client

```
# Use CouchDB to create a CouchDB client
# from cloudant.client import CouchDB
# client = CouchDB(USERNAME, PASSWORD, url='http://127.0.0.1:5984', connect=True)

# Use Cloudant to create a Cloudant client using account
from cloudant.client import Cloudant
client = Cloudant(USERNAME, PASSWORD, account=ACCOUNT_NAME, connect=True)
# or using url
# client = Cloudant(USERNAME, PASSWORD, url='https://acct.cloudant.com')
```

(continues on next page)

(continued from previous page)

```
# or with a 429 replay adapter that includes configured retries and initial backoff
# client = Cloudant(USERNAME, PASSWORD, account=ACCOUNT_NAME,
#                   adapter=Replay429Adapter(retries=10, initialBackoff=0.01))

# or with a connect and read timeout of 5 minutes
# client = Cloudant(USERNAME, PASSWORD, account=ACCOUNT_NAME,
#                   timeout=300)

# Perform client tasks...
session = client.session()
print('Username: {0}'.format(session['userCtx']['name']))
print('Databases: {0}'.format(client.all_dbs()))

# Disconnect from the server
client.disconnect()
```

## 2.2 Authentication

When constructing a `Cloudant` client, you can authenticate using the cookie authentication functionality. The server will always attempt to automatically renew the cookie shortly before its expiry. However, if the client does not send a request to the server during this renewal window and `auto_renew=False` then the cookie is not renewed.

Using `auto_renew=True` will attempt to renew the cookie at any point during the lifetime of the session when either of the following statements hold true:

- The server returns a `credentials_expired` error message.
- The server returns a `401 Unauthorized` status code.
- The server returns a `403 Forbidden` status code.

```
# Create client using auto_renew to automatically renew expired cookie auth
client = Cloudant(USERNAME, PASSWORD, url='https://acct.cloudant.com',
                  connect=True,
                  auto_renew=True)
```

## 2.3 Identity and Access Management (IAM)

IBM Cloud Identity & Access Management enables you to securely authenticate users and control access to all cloud resources consistently in the IBM Bluemix Cloud Platform.

See IBM Cloud Identity and Access Management for more information.

The production IAM token service at *https://iam.cloud.ibm.com/identity/token* is used by default. You can set an `IAM_TOKEN_URL` environment variable to override this.

You can easily connect to your Cloudant account using an IAM API key:

```
# Authenticate using an IAM API key
client = Cloudant.iam(ACCOUNT_NAME, API_KEY, connect=True)
```

## 2.4 Resource sharing

The `Cloudant` or `CouchDB` client objects make HTTP calls using the `requests` library. `requests` uses the urllib3 library which features connection pooling and thread safety.

Connection pools can be managed by using the `requests` library's HTTPAdapter when constructing a `Cloudant` or `ClouchDB` client instance. The default number set by the `urllib3` library for cached connection pools is 10. Use the `HTTPAdapter` argument `pool_connections` to set the number of urllib3 connection pools to cache, and the `pool_maxsize` argument to set the maximum number of connections to save in the pool.

Although the `client` session is documented as thread safe and it's possible for a static `client` to be accessible by multiple threads, there are still cases that do not guarantee thread safe execution. It's recommended to use one `client` object per thread.

```
# Create client with 15 cached pool connections and a max pool size of 100
httpAdapter = HTTPAdapter(pool_connections=15, pool_maxsize=100)
client = Cloudant(USERNAME, PASSWORD, url='https://acct.cloudant.com'
                  connect=True,
                  adapter=httpAdapter)
```

Note: Idle connections within the pool may be terminated by the server, so will not remain open indefinitely meaning that this will not completely remove the overhead of creating new connections.

### 2.4.1 Using library in app server environment

This library can be used in an app server, and the example below shows how to use `client` in a `flask` app server.

```
from flask import Flask
import atexit

app = Flask(__name__)

@app.route('/')
def hello_world():
    # Cookie authentication can be renewed automatically using ``auto_renew=True``
    # which is typically what you would require when running in an application
    # server where the connection may stay open for a long period of time

    # Note: Each time you instantiate an instance of the Cloudant client, an
    # authentication request will be made to Cloudant to retrieve the session cookie.
    # If the performance overhead of this call is a concern for you, consider
    # using vanilla python requests with a custom subclass of HTTPAdapter that
    # performs the authentication call to Cloudant when it establishes the http
    # connection during the creation of the connection pool.
    client = Cloudant(USERNAME, PASSWORD, url='https://acct.cloudant.com',
                      connect=True,
                      auto_renew=True)

    # do something with client
    return 'Hello World!'

# When shutting down the app server, use ``client.disconnect()`` to properly
# logout and end the ``client`` session
@atexit.register
def shutdown():
    client.disconnect()
```

## 2.5 Databases

Once a connection is established you can then create a database, open an existing database, or delete a database. The following examples assume a client connection has already been established.

### 2.5.1 Creating a database

```
# Create a database using an initialized client
# The result is a new CloudantDatabase or CouchDatabase based on the client
my_database = client.create_database('my_database')

# You can check that the database exists
if my_database.exists():
    print('SUCCESS!!')
```

### 2.5.2 Opening a database

Opening an existing database is done by supplying the name of an existing database to the client. Since the `Cloudant` and `CouchDB` classes are sub-classes of `dict`, this can be accomplished through standard Python `dict` notation.

```
# Open an existing database
my_database = client['my_database']
```

### 2.5.3 Deleting a database

```
# Delete a database using an initialized client
client.delete_database('my_database')
```

## 2.6 Documents

Working with documents using this library is handled through the use of Document objects and Database API methods. A document context manager is also provided to simplify the process. This is discussed later in the *Context managers* section. The examples that follow demonstrate how to create, read, update, and delete a document. These examples assume that either a CloudantDatabase or a CouchDatabase object already exists.

### 2.6.1 Creating a document

```
# Create document content data
data = {
    '_id': 'julia30', # Setting _id is optional
    'name': 'Julia',
    'age': 30,
    'pets': ['cat', 'dog', 'frog']
    }

# Create a document using the Database API
my_document = my_database.create_document(data)
```

(continues on next page)

(continued from previous page)

```
# Check that the document exists in the database
if my_document.exists():
    print('SUCCESS!!')
```

### 2.6.2 Retrieving a document

Accessing a document from a database is done by supplying the document identifier of an existing document to either a `CloudantDatabase` or a `CouchDatabase` object. Since the `CloudantDatabase` and `CouchDatabase` classes are sub-classes of `dict`, this is accomplished through standard `dict` notation.

```
my_document = my_database['julia30']

# Display the document
print(my_document)
```

### 2.6.3 Checking if a document exists

You can check if a document exists in a database the same way you would check if a `dict` has a key-value pair by key.

```
doc_exists = 'julia30' in my_database

if doc_exists:
    print('document with _id julia30 exists')
```

### 2.6.4 Retrieve all documents

You can also iterate over a `CloudantDatabase` or a `CouchDatabase` object to retrieve all documents in a database.

```
# Get all of the documents from my_database
for document in my_database:
    print(document)
```

### 2.6.5 Update a document

```
# First retrieve the document
my_document = my_database['julia30']

# Update the document content
# This can be done as you would any other dictionary
my_document['name'] = 'Jules'
my_document['age'] = 6

# You must save the document in order to update it on the database
my_document.save()
```

### 2.6.6 Delete a document

```
# First retrieve the document
my_document = my_database['julia30']

# Delete the document
my_document.delete()
```

## 2.7 Dealing with results

If you want to get Pythonic with your returned data content, we've added a `Result` class that provides a key accessible, sliceable, and iterable interface to result collections. To use it, construct a `Result` object passing in a reference to a raw data callable such as the `all_docs` method from a database object or a `view` object itself, which happens to be defined as callable and then access the data as you would using standard Python key access, slicing, and iteration techniques. The following set of examples illustrate `Result` key access, slicing and iteration over a result collection in action. It assumes that either a `CloudantDatabase` or a `CouchDatabase` object already exists.

```
from cloudant.result import Result, ResultByKey

# Retrieve Result wrapped document content.
# Note: The include_docs parameter is optional and is used to illustrate that view␣
↪query
# parameters can be used to customize the result collection.
result_collection = Result(my_database.all_docs, include_docs=True)

# Get the result at a given location in the result collection
# Note: Valid result collection indexing starts at 0
result = result_collection[0]                    # result is the 1st in the collection
result = result_collection[9]                    # result is the 10th in the collection

# Get the result for matching a key
result = result_collection['julia30']            # result is all that match key
↪'julia30'

# If your key is an integer then use the ResultByKey class to differentiate your␣
↪integer
# key from an indexed location within the result collection which is also an integer.
result = result_collection[ResultByKey(9)]       # result is all that match key 9

# Slice by key values
result = result_collection['julia30': 'ruby99'] # result is between and including keys
result = result_collection['julia30': ]          # result is after and including key
result = result_collection[: 'ruby99']           # result is up to and including key

# Slice by index values
result = result_collection[100: 200]             # result is between 100 to 200,␣
↪including 200th
result = result_collection[: 200]                # result is up to and including the␣
↪200th
result = result_collection[100: ]                # result is after the 100th

# Iterate over the result collection
for result in result_collection:
    print(result)
```

## 2.8 Context managers

Now that we've gone through the basics, let's take a look at how to simplify the process of connection, database acquisition, and document management through the use of Python *with* blocks and this library's context managers.

Handling your business using *with* blocks saves you from having to connect and disconnect your client as well as saves you from having to perform a lot of fetch and save operations as the context managers handle these operations for you.

This example uses the `cloudant` context helper to illustrate the process but identical functionality exists for CouchDB through the `couchdb` and `couchdb_admin_party` context helpers.

```
from cloudant import cloudant

# ...or use CouchDB variant
# from cloudant import couchdb

# Perform a connect upon entry and a disconnect upon exit of the block
with cloudant(USERNAME, PASSWORD, account=ACCOUNT_NAME) as client:

# ...or use CouchDB variant
# with couchdb(USERNAME, PASSWORD, url=COUCHDB_URL) as client:

    # Perform client tasks...
    session = client.session()
    print('Username: {0}'.format(session['userCtx']['name']))
    print('Databases: {0}'.format(client.all_dbs()))

    # Create a database
    my_database = client.create_database('my_database')
    if my_database.exists():
        print('SUCCESS!!')

    # You can open an existing database
    del my_database
    my_database = client['my_database']
```

The following example uses the `Document` context manager. Here we make multiple updates to a single document. Note that we don't save to the server after each update. We only save once to the server upon exiting the `Document` context manager.

```
from cloudant import cloudant
from cloudant.document import Document

with cloudant(USERNAME, PASSWORD, account=ACCOUNT_NAME) as client:

    my_database = client.create_database('my_database')

    # Upon entry into the document context, fetches the document from the
    # remote database, if it exists. Upon exit from the context, saves the
    # document to the remote database with changes made within the context
    # or creates a new document.
    with Document(database, 'julia006') as document:
        # If document exists, it's fetched from the remote database
        # Changes are made locally
        document['name'] = 'Julia'
        document['age'] = 6
        # The document is saved to the remote database
```

(continues on next page)

(continued from previous page)

```
    # Display a Document
    print(my_database['julia30'])

    # Delete the database
    client.delete_database('my_database')

    print('Databases: {0}'.format(client.all_dbs()))
```

Always use the `_deleted` document property to delete a document from within a `Document` context manager. For example:

```
with Document(my_database, 'julia30') as doc:
    doc['_deleted'] = True
```

*You can also delete non underscore prefixed document keys to reduce the size of the request.*

**Warning:** Don't use the `doc.delete()` method inside your `Document` context manager. This method immediately deletes the document on the server and clears the local document dictionary. A new, empty document is still saved to the server upon exiting the context manager.

## 2.9 Endpoint access

If for some reason you need to call a Cloudant/CouchDB endpoint directly rather using the API you can still benefit from the Cloudant/CouchDB client's authentication and session usage by directly accessing its underlying Requests session object.

Access the session object using the `r_session` attribute on your client object. From there, use the session to make requests as the user the client is set up with. The following example shows a `GET` to the `_all_docs` endpoint, but obviously you can use this for any HTTP request to the Cloudant/CouchDB server. This example assumes that either a `Cloudant` or a `CouchDB` client object already exists.

```
# Define the end point and parameters
end_point = '{0}/{1}'.format(client.server_url, 'my_database/_all_docs')
params = {'include_docs': 'true'}

# Issue the request
response = client.r_session.get(end_point, params=params)

# Display the response content
print(response.json())
```

## 2.10 TLS 1.2 Support

The TLS protocol is used to encrypt communications across a network to ensure that transmitted data remains private. There are three released versions of TLS: 1.0, 1.1, and 1.2. All HTTPS connections use TLS.

If your server enforces the use of TLS 1.2 then the python-cloudant client will continue to work as expected (assuming you're running a version of Python/OpenSSL that supports TLS 1.2).

# CHAPTER 3

# Cloudant client library API

Cloudant / CouchDB Python client library API package

cloudant.**cloudant**(**args*, ***kwds*)

Provides a context manager to create a Cloudant session and provide access to databases, docs etc.

**Parameters**

- **user** (*str*) – Username used to connect to Cloudant.
- **passwd** (*str*) – Authentication token used to connect to Cloudant.
- **account** (*str*) – The Cloudant account name. If the account parameter is present, it will be used to construct the Cloudant service URL.
- **url** (*str*) – If the account is not present and the url parameter is present then it will be used to set the Cloudant service URL. The url must be a fully qualified http/https URL.
- **x_cloudant_user** (*str*) – Override the X-Cloudant-User setting used to authenticate. This is needed to authenticate on one's behalf, eg with an admin account. This parameter must be accompanied by the url parameter. If the url parameter is omitted then the x_cloudant_user parameter setting is ignored.
- **encoder** (*str*) – Optional json Encoder object used to encode documents for storage. Defaults to json.JSONEncoder.

For example:

```
# cloudant context manager
from cloudant import cloudant

with cloudant(USERNAME, PASSWORD, account=ACCOUNT_NAME) as client:
    # Context handles connect() and disconnect() for you.
    # Perform library operations within this context.  Such as:
    print client.all_dbs()
    # ...
```

cloudant.**cloudant_bluemix**(**args*, ***kwds*)

Provides a context manager to create a Cloudant session and provide access to databases, docs etc.

**Parameters**

- **vcap_services** (*dict or str*) – VCAP_SERVICES environment variable
- **instance_name** (*str*) – Optional Bluemix instance name. Only required if multiple Cloudant instances are available.
- **service_name** (*str*) – Optional Bluemix service name.
- **encoder** (*str*) – Optional json Encoder object used to encode documents for storage. Defaults to json.JSONEncoder.

Loads all configuration from the specified VCAP_SERVICES Cloud Foundry environment variable. The VCAP_SERVICES variable contains connection information to access a service instance. For example:

```
{
    "VCAP_SERVICES": {
        "cloudantNoSQLDB": [
            {
                "credentials": {
                    "apikey": "some123api456key"
                    "username": "example",
                    "password": "xxxxxxx",
                    "host": "example.cloudant.com",
                    "port": 443,
                    "url": "https://example:xxxxxxx@example.cloudant.com"
                },
                "syslog_drain_url": null,
                "label": "cloudantNoSQLDB",
                "provider": null,
                "plan": "Lite",
                "name": "Cloudant NoSQL DB"
            }
        ]
    }
}
```

See Cloud Foundry Environment Variables.

Example usage:

```
import os

# cloudant_bluemix context manager
from cloudant import cloudant_bluemix

with cloudant_bluemix(os.getenv('VCAP_SERVICES'), 'Cloudant NoSQL DB') as client:
    # Context handles connect() and disconnect() for you.
    # Perform library operations within this context.  Such as:
    print client.all_dbs()
    # ...
```

cloudant.**cloudant_iam**(**args*, ***kwds*)

Provides a context manager to create a Cloudant session using IAM authentication and provide access to databases, docs etc.

**Parameters**

- **account_name** – Cloudant account name.
- **api_key** – IAM authentication API key.

For example:

```
# cloudant context manager
from cloudant import cloudant_iam

with cloudant_iam(ACCOUNT_NAME, API_KEY) as client:
    # Context handles connect() and disconnect() for you.
    # Perform library operations within this context.  Such as:
    print client.all_dbs()
    # ...
```

cloudant.**couchdb**(**args*, ***kwds*)

Provides a context manager to create a CouchDB session and provide access to databases, docs etc.

**Parameters**

- **user** (*str*) – Username used to connect to CouchDB.
- **passwd** (*str*) – Passcode used to connect to CouchDB.
- **url** (*str*) – URL for CouchDB server.
- **encoder** (*str*) – Optional json Encoder object used to encode documents for storage. Defaults to json.JSONEncoder.

For example:

```
# couchdb context manager
from cloudant import couchdb

with couchdb(USERNAME, PASSWORD, url=COUCHDB_URL) as client:
    # Context handles connect() and disconnect() for you.
    # Perform library operations within this context.  Such as:
    print client.all_dbs()
    # ...
```

cloudant.**couchdb_admin_party**(**args*, ***kwds*)

Provides a context manager to create a CouchDB session in Admin Party mode and provide access to databases, docs etc.

**Parameters**

- **url** (*str*) – URL for CouchDB server.
- **encoder** (*str*) – Optional json Encoder object used to encode documents for storage. Defaults to json.JSONEncoder.

For example:

```
# couchdb_admin_party context manager
from cloudant import couchdb_admin_party

with couchdb_admin_party(url=COUCHDB_URL) as client:
    # Context handles connect() and disconnect() for you.
    # Perform library operations within this context.  Such as:
    print client.all_dbs()
    # ...
```

## 3.1 Modules

### 3.1.1 client

Top level API module that maps to a Cloudant or CouchDB client connection instance.

**class** cloudant.client.**Cloudant**(*cloudant_user*, *auth_token*, ***kwargs*)
Bases: *cloudant.client.CouchDB*

Encapsulates a Cloudant client, handling top level user API calls having to do with session and database management.

Maintains a requests.Session for working with the instance specified in the constructor.

Parameters can be passed in to control behavior:

**Parameters**

- **cloudant_user** (*str*) – Username used to connect to Cloudant.
- **auth_token** (*str*) – Authentication token used to connect to Cloudant.
- **account** (*str*) – The Cloudant account name. If the account parameter is present, it will be used to construct the Cloudant service URL.
- **url** (*str*) – If the account is not present and the url parameter is present then it will be used to set the Cloudant service URL. The url must be a fully qualified http/https URL.
- **x_cloudant_user** (*str*) – Override the X-Cloudant-User setting used to authenticate. This is needed to authenticate on one's behalf, eg with an admin account. This parameter must be accompanied by the url parameter. If the url parameter is omitted then the x_cloudant_user parameter setting is ignored.
- **encoder** (*str*) – Optional json Encoder object used to encode documents for storage. Defaults to json.JSONEncoder.
- **adapter** (*requests.HTTPAdapter*) – Optional adapter to use for configuring requests.

**bill**(*year=None*, *month=None*)
Retrieves Cloudant billing data, optionally for a given year and month.

**Parameters**

- **year** (*int*) – Year to query against, for example 2014. Optional parameter. Defaults to None. If used, it must be accompanied by `month`.
- **month** (*int*) – Month to query against that must be an integer between 1 and 12. Optional parameter. Defaults to None. If used, it must be accompanied by `year`.

**Returns** Billing data in JSON format

**classmethod bluemix**(*vcap_services*, *instance_name=None*, *service_name=None*, ***kwargs*)
Create a Cloudant session using a VCAP_SERVICES environment variable.

**Parameters**

- **vcap_services** (*dict or str*) – VCAP_SERVICES environment variable
- **instance_name** (*str*) – Optional Bluemix instance name. Only required if multiple Cloudant instances are available.
- **service_name** (*str*) – Optional Bluemix service name.

Example usage:

```
import os
from cloudant.client import Cloudant

client = Cloudant.bluemix(os.getenv('VCAP_SERVICES'),
                          'Cloudant NoSQL DB')

print client.all_dbs()
```

**cors_configuration**()
Retrieves the current CORS configuration.

**Returns** CORS data in JSON format

**cors_origins**()
Retrieves a list of CORS origins.

**Returns** List of CORS origins

**db_updates**(*raw_data=False*, ***kwargs*)
Returns the _db_updates feed iterator. The _db_updates feed can be iterated over and once complete can also provide the last sequence identifier of the feed. If necessary, the iteration can be stopped by issuing a call to the stop() method on the returned iterator object.

For example:

```
# Iterate over a "normal" _db_updates feed
db_updates = client.db_updates()
for db_update in db_updates:
    print(db_update)
print(db_updates.last_seq)

# Iterate over a "continuous" _db_updates feed with additional options
db_updates = client.db_updates(feed='continuous', since='now',␣
↪descending=True)
for db_update in db_updates:
    if some_condition:
        db_updates.stop()
    print(db_update)
```

**Parameters**

- **raw_data** (*bool*) – If set to True then the raw response data will be streamed otherwise if set to False then JSON formatted data will be streamed. Default is False.
- **descending** (*bool*) – Whether results should be returned in descending order, i.e. the latest event first. By default, the oldest event is returned first.
- **feed** (*str*) – Type of feed. Valid values are continuous, longpoll, and normal. Default is normal.
- **heartbeat** (*int*) – Time in milliseconds after which an empty line is sent during longpoll or continuous if there have been no changes. Must be a positive number. Default is no heartbeat.
- **limit** (*int*) – Maximum number of rows to return. Must be a positive number. Default is no limit.
- **since** – Start the results from changes after the specified sequence identifier. In other words, using since excludes from the list all changes up to and including the specified

sequence identifier. If since is 0 (the default), or omitted, the request returns all changes. If it is `now`, only changes made after the time of the request will be emitted.

- **timeout** (*int*) – Number of milliseconds to wait for data before terminating the response. `heartbeat` supersedes `timeout` if both are supplied.
- **chunk_size** (*int*) – The HTTP response stream chunk size. Defaults to 512.

**Returns** Feed object that can be iterated over as a `_db_updates` feed.

**disable_cors**()
Switches CORS off.

**Returns** CORS status in JSON format

**generate_api_key**()
Creates and returns a new API Key/pass pair.

**Returns** API key/pass pair in JSON format

**classmethod iam**(*account_name*, *api_key*, ***kwargs*)
Create a Cloudant client that uses IAM authentication.

**Parameters**

- **account_name** – Cloudant account name.
- **api_key** – IAM authentication API key.

**infinite_db_updates**(***kwargs*)
Returns an infinite (perpetually refreshed) `_db_updates` feed iterator. If necessary, the iteration can be stopped by issuing a call to the `stop()` method on the returned iterator object.

For example:

```
# Iterate over an infinite _db_updates feed
db_updates = client.infinite_db_updates()
for db_update in db_updates:
    if some_condition:
        db_updates.stop()
    print(db_update)
```

**Parameters**

- **descending** (*bool*) – Whether results should be returned in descending order, i.e. the latest event first. By default, the oldest event is returned first.
- **heartbeat** (*int*) – Time in milliseconds after which an empty line is sent if there have been no changes. Must be a positive number. Default is no heartbeat.
- **since** – Start the results from changes after the specified sequence identifier. In other words, using since excludes from the list all changes up to and including the specified sequence identifier. If since is 0 (the default), or omitted, the request returns all changes. If it is `now`, only changes made after the time of the request will be emitted.
- **timeout** (*int*) – Number of milliseconds to wait for data before terminating the response. `heartbeat` supersedes `timeout` if both are supplied.
- **chunk_size** (*int*) – The HTTP response stream chunk size. Defaults to 512.

**Returns** Feed object that can be iterated over as a `_db_updates` feed.

**requests_usage**(*year=None*, *month=None*)
Retrieves Cloudant requests usage data, optionally for a given year and month.

**Parameters**

- **year** (*int*) – Year to query against, for example 2014. Optional parameter. Defaults to None. If used, it must be accompanied by `month`.
- **month** (*int*) – Month to query against that must be an integer between 1 and 12. Optional parameter. Defaults to None. If used, it must be accompanied by `year`.

**Returns** Requests usage data in JSON format

**shared_databases**()
Retrieves a list containing the names of databases shared with this account.

**Returns** List of database names

**update_cors_configuration**(*enable_cors=True*, *allow_credentials=True*, *origins=None*, *overwrite_origins=False*)
Merges existing CORS configuration with updated values.

**Parameters**

- **enable_cors** (*bool*) – Enables/disables CORS. Defaults to True.
- **allow_credentials** (*bool*) – Allows authentication credentials. Defaults to True.
- **origins** (*list*) – List of allowed CORS origin(s). Special cases are a list containing a single "*" which will allow any origin and an empty list which will not allow any origin. Defaults to None.
- **overwrite_origins** (*bool*) – Dictates whether the origins list is overwritten of appended to. Defaults to False.

**Returns** CORS configuration update status in JSON format

**volume_usage**(*year=None*, *month=None*)
Retrieves Cloudant volume usage data, optionally for a given year and month.

**Parameters**

- **year** (*int*) – Year to query against, for example 2014. Optional parameter. Defaults to None. If used, it must be accompanied by `month`.
- **month** (*int*) – Month to query against that must be an integer between 1 and 12. Optional parameter. Defaults to None. If used, it must be accompanied by `year`.

**Returns** Volume usage data in JSON format

**class** cloudant.client.**CouchDB**(*user*, *auth_token*, *admin_party=False*, ***kwargs*)
Bases: dict

Encapsulates a CouchDB client, handling top level user API calls having to do with session and database management.

Maintains a requests.Session for working with the instance specified in the constructor.

Parameters can be passed in to control behavior:

**Parameters**

- **user** (*str*) – Username used to connect to CouchDB.
- **auth_token** (*str*) – Authentication token used to connect to CouchDB.

- **admin_party** (*bool*) – Setting to allow the use of Admin Party mode in CouchDB. Defaults to `False`.
- **url** (*str*) – URL for CouchDB server.
- **encoder** (*str*) – Optional json Encoder object used to encode documents for storage. Defaults to json.JSONEncoder.
- **adapter** (*requests.HTTPAdapter*) – Optional adapter to use for configuring requests.
- **connect** (*bool*) – Keyword argument, if set to True performs the call to connect as part of client construction. Default is False.
- **auto_renew** (*bool*) – Keyword argument, if set to True performs automatic renewal of expired session authentication settings. Default is False.
- **timeout** (*float*) – Timeout in seconds (use float for milliseconds, for example 0.1 for 100 ms) for connecting to and reading bytes from the server. If a single value is provided it will be applied to both the connect and read timeouts. To specify different values for each timeout use a tuple. For example, a 10 second connect timeout and a 1 minute read timeout would be (10, 60). This follows the same behaviour as the Requests library timeout argument. but will apply to every request made using this client.
- **use_basic_auth** (*bool*) – Keyword argument, if set to True performs basic access authentication with server. Default is False.
- **use_iam** (*bool*) – Keyword argument, if set to True performs IAM authentication with server. Default is False. Use `iam()` to construct an IAM authenticated client.
- **iam_client_id** (*string*) – Keyword argument, client ID to use when authenticating with the IAM token server. Default is `None`.
- **iam_client_secret** (*string*) – Keyword argument, client secret to use when authenticating with the IAM token server. Default is `None`.

**__delitem__**(*key*, *remote=False*)

Overrides dictionary __delitem__ behavior to make deleting the database key a proxy for deleting the database. If remote=True then it will delete the database on the remote server, otherwise only the local cached object will be removed.

**Parameters**

- **key** (*str*) – Database name of the database to be deleted.
- **remote** (*bool*) – Dictates whether the locally cached database is deleted or a remote request is made to delete the database from the server. Defaults to False.

**__getitem__**(*key*)

Overrides dictionary __getitem__ behavior to provide a database instance for the specified key.

If the database instance does not exist locally, then a remote request is made and the database is subsequently added to the local cache and returned to the caller.

If the database instance already exists locally then it is returned and a remote request is not performed.

A KeyError will result if the database does not exist locally or on the server.

**Parameters** **key** (*str*) – Database name used to retrieve the database object.

**Returns** Database object

**__setitem__**(*key*, *value*, *remote=False*)
Override dictionary __setitem__ behavior to verify that only database instances are added as keys. If remote=True then also create the database remotely if the database does not exist.

Note: The only way to override the default for the `remote` argument setting it to True is to call __setitem__ directly. A much simpler approach is to use *`create_database()`* instead, if your intention is to create a database remotely.

**Parameters**

- **key** (*str*) – Database name to be used as the key for the database in the locally cached dictionary.
- **value** – Database object to be used in the locally cached dictionary.
- **remote** (*bool*) – Dictates whether the method will attempt to create the database remotely or not. Defaults to False.

**all_dbs**()
Retrieves a list of all database names for the current client.

**Returns** List of database names for the client

**basic_auth_str**()
Composes a basic http auth string, suitable for use with the _replicator database, and other places that need it.

**Returns** Basic http authentication string

**change_credentials**(*user=None*, *auth_token=None*)
Change login credentials.

**Parameters**

- **user** (*str*) – Username used to connect to server.
- **auth_token** (*str*) – Authentication token used to connect to server.

**connect**()
Starts up an authentication session for the client using cookie authentication if necessary.

**create_database**(*dbname*, ***kwargs*)
Creates a new database on the remote server with the name provided and adds the new database object to the client's locally cached dictionary before returning it to the caller. The method will optionally throw a CloudantClientException if the database exists remotely.

**Parameters**

- **dbname** (*str*) – Name used to create the database.
- **throw_on_exists** (*bool*) – Boolean flag dictating whether or not to throw a CloudantClientException when attempting to create a database that already exists.

**Returns** The newly created database object

**db_updates**(*raw_data=False*, ***kwargs*)
Returns the `_db_updates` feed iterator. While iterating over the feed, if necessary, the iteration can be stopped by issuing a call to the `stop()` method on the returned iterator object.

For example:

```
# Iterate over a "longpoll" _db_updates feed
db_updates = client.db_updates()
for db_update in db_updates:
```

(continues on next page)

(continued from previous page)

```
    if some_condition:
        db_updates.stop()
    print(db_update)

# Iterate over a "continuous" _db_updates feed with additional options
db_updates = client.db_updates(feed='continuous', heartbeat=False)
for db_update in db_updates:
    if some_condition:
        db_updates.stop()
    print(db_update)
```

**Parameters**

- **raw_data** (*bool*) – If set to True then the raw response data will be streamed otherwise if set to False then JSON formatted data will be streamed. Default is False.
- **feed** (*str*) – Type of feed. Valid values are `continuous`, and `longpoll`. Default is `longpoll`.
- **heartbeat** (*bool*) – Whether CouchDB will send a newline character on timeout. Default is True.
- **timeout** (*int*) – Number of seconds to wait for data before terminating the response.
- **chunk_size** (*int*) – The HTTP response stream chunk size. Defaults to 512.

**Returns** Feed object that can be iterated over as a `_db_updates` feed.

**delete_database**(*dbname*)

Removes the named database remotely and locally. The method will throw a CloudantClientException if the database does not exist.

**Parameters** **dbname** (*str*) – Name of the database to delete.

**disconnect**()

Ends a client authentication session, performs a logout and a clean up.

**features**()

lazy fetch and cache features

**get**(*key*, *default=None*, *remote=False*)

Overrides dictionary get behavior to retrieve database objects with support for returning a default. If remote=True then a remote request is made to retrieve the database from the remote server, otherwise the client's locally cached database object is returned.

**Parameters**

- **key** (*str*) – Database name used to retrieve the database object.
- **default** (*str*) – Default database name. Defaults to None.
- **remote** (*bool*) – Dictates whether the locally cached database is returned or a remote request is made to retrieve the database from the server. Defaults to False.

**Returns** Database object

**is_iam_authenticated**

Show if a client has authenticated using an IAM API key.

**Returns** True if client is IAM authenticated. False otherwise.

**keys**(*remote=False*)
Returns the database names for this client. Default is to return only the locally cached database names, specify `remote=True` to make a remote request to include all databases.

**Parameters** **remote** (*bool*) – Dictates whether the list of locally cached database names are returned or a remote request is made to include an up to date list of databases from the server. Defaults to False.

**Returns** List of database names

**metadata**()
Retrieves the remote server metadata dictionary.

**Returns** Dictionary containing server metadata details

**session**()
Retrieves information about the current login session to verify data related to sign in.

**Returns** Dictionary of session info for the current session.

**session_cookie**()
Retrieves the current session cookie.

**Returns** Session cookie for the current session

**session_login**(*user=None*, *passwd=None*)
Performs a session login by posting the auth information to the _session endpoint.

**Parameters**

- **user** (*str*) – Username used to connect to server.
- **auth_token** (*str*) – Authentication token used to connect to server.

**session_logout**()
Performs a session logout and clears the current session by sending a delete request to the _session endpoint.

### 3.1.2 database

API module that maps to a Cloudant or CouchDB database instance.

**class** cloudant.database.**CloudantDatabase**(*client*, *database_name*, *fetch_limit=100*)
Bases: *cloudant.database.CouchDatabase*

Encapsulates a Cloudant database. A CloudantDatabase object is instantiated with a reference to a client/session. It supports accessing the documents, and various database features such as the document indexes, changes feed, design documents, etc.

**Parameters**

- **client** (Cloudant) – Client instance used by the database.
- **database_name** (*str*) – Database name used to reference the database.
- **fetch_limit** (*int*) – Optional fetch limit used to set the max number of documents to fetch per query during iteration cycles. Defaults to 100.

**get_search_result**(*ddoc_id*, *index_name*, ***query_params*)
Retrieves the raw JSON content from the remote database based on the search index on the server, using the query_params provided as query parameters. A `query` parameter containing the Lucene query syntax is mandatory.

Example for search queries:

```
# Assuming that 'searchindex001' exists as part of the
# 'ddoc001' design document in the remote database...
# Retrieve documents where the Lucene field name is 'name' and
# the value is 'julia*'
resp = db.get_search_result('ddoc001', 'searchindex001',
                            query='name:julia*',
                            include_docs=True)
for row in resp['rows']:
    # Process search index data (in JSON format).
```

Example if the search query requires grouping by using the `group_field` parameter:

```
# Assuming that 'searchindex001' exists as part of the
# 'ddoc001' design document in the remote database...
# Retrieve JSON response content, limiting response to 10 documents
resp = db.get_search_result('ddoc001', 'searchindex001',
                            query='name:julia*',
                            group_field='name',
                            limit=10)
for group in resp['groups']:
    for row in group['rows']:
    # Process search index data (in JSON format).
```

**Parameters**

- **ddoc_id** (*str*) – Design document id used to get the search result.
- **index_name** (*str*) – Name used in part to identify the index.
- **bookmark** (*str*) – Optional string that enables you to specify which page of results you require. Only valid for queries that do not specify the `group_field` query parameter.
- **counts** (*list*) – Optional JSON array of field names for which counts should be produced. The response will contain counts for each unique value of this field name among the documents matching the search query. Requires the index to have faceting enabled.
- **drilldown** (*list*) – Optional list of fields that each define a pair of a field name and a value. This field can be used several times. The search will only match documents that have the given value in the field name. It differs from using `query=fieldname:value` only in that the values are not analyzed.
- **group_field** (*str*) – Optional string field by which to group search matches. Fields containing other data (numbers, objects, arrays) can not be used.
- **group_limit** (*int*) – Optional number with the maximum group count. This field can only be used if `group_field` query parameter is specified.
- **group_sort** – Optional JSON field that defines the order of the groups in a search using `group_field`. The default sort order is relevance. This field can have the same values as the sort field, so single fields as well as arrays of fields are supported.
- **limit** (*int*) – Optional number to limit the maximum count of the returned documents. In case of a grouped search, this parameter limits the number of documents per group.
- **query/q** – A Lucene query in the form of `name:value`. If name is omitted, the special value `default` is used. The `query` parameter can be abbreviated as `q`.
- **ranges** – Optional JSON facet syntax that reuses the standard Lucene syntax to return counts of results which fit into each specified category. Inclusive range queries are denoted by brackets. Exclusive range queries are denoted by curly brackets. For example

`ranges={"price":{"cheap":"[0 TO 100]"}}` has an inclusive range of 0 to 100. Requires the index to have faceting enabled.

- **sort** – Optional JSON string of the form `fieldname<type>` for ascending or `-fieldname<type>` for descending sort order. Fieldname is the name of a string or number field and type is either number or string or a JSON array of such strings. The type part is optional and defaults to number.
- **stale** (*str*) – Optional string to allow the results from a stale index to be used. This makes the request return immediately, even if the index has not been completely built yet.
- **highlight_fields** (*list*) – Optional list of fields which should be highlighted.
- **highlight_pre_tag** (*str*) – Optional string inserted before the highlighted word in the highlights output. Defaults to `<em>`.
- **highlight_post_tag** (*str*) – Optional string inserted after the highlighted word in the highlights output. Defaults to `</em>`.
- **highlight_number** (*int*) – Optional number of fragments returned in highlights. If the search term occurs less often than the number of fragments specified, longer fragments are returned. Default is 1.
- **highlight_size** (*int*) – Optional number of characters in each fragment for highlights. Defaults to 100 characters.
- **include_fields** (*list*) – Optional list of field names to include in search results. Any fields included must have been indexed with the `store:true` option.

**Returns** Search query result data in JSON format

**security_document**()
Retrieves the security document for the current database containing information about the users that the database is shared with.

**Returns** Security document as a `dict`

**security_url**
Constructs and returns the security document URL.

**Returns** Security document URL

**shards**()
Retrieves information about the shards in the current remote database.

**Returns** Shard information retrieval status in JSON format

**share_database**(*username*, *roles=None*)
Shares the current remote database with the username provided. You can grant varying degrees of access rights, default is to share read-only, but additional roles can be added by providing the specific roles as a `list` argument. If the user already has this database shared with them then it will modify/overwrite the existing permissions.

**Parameters**

- **username** (*str*) – Cloudant user to share the database with.
- **roles** (*list*) – A list of roles to grant to the named user.

**Returns** Share database status in JSON format

**unshare_database**(*username*)
Removes all sharing with the named user for the current remote database. This will remove the entry

for the user from the security document. To modify permissions, use the *share_database()* method instead.

**Parameters** **username** (*str*) – Cloudant user to unshare the database from.

**Returns** Unshare database status in JSON format

**class** cloudant.database.**CouchDatabase**(*client*, *database_name*, *fetch_limit=100*)
Bases: dict

Encapsulates a CouchDB database. A CouchDatabase object is instantiated with a reference to a client/session. It supports accessing the documents, and various database features such as the document indexes, changes feed, design documents, etc.

**Parameters**

- **client** (CouchDB) – Client instance used by the database.
- **database_name** (*str*) – Database name used to reference the database.
- **fetch_limit** (*int*) – Optional fetch limit used to set the max number of documents to fetch per query during iteration cycles. Defaults to 100.

**__getitem__**(*key*)
Overrides dictionary __getitem__ behavior to provide a document instance for the specified key from the current database.

If the document instance does not exist locally, then a remote request is made and the document is subsequently added to the local cache and returned to the caller.

If the document instance already exists locally then it is returned and a remote request is not performed.

A KeyError will result if the document does not exist locally or in the remote database.

**Parameters** **key** (*str*) – Document id used to retrieve the document from the database.

**Returns** A Document or DesignDocument object depending on the specified document id (key)

**__iter__**(*remote=True*)
Overrides dictionary __iter__ behavior to provide iterable Document results. By default, Documents are fetched from the remote database, in batches equal to the database object's defined `fetch_limit`, yielding Document/DesignDocument objects.

If `remote=False` then the locally cached Document objects are iterated over with no attempt to retrieve documents from the remote database.

**Parameters** **remote** (*bool*) – Dictates whether the locally cached Document objects are returned or a remote request is made to retrieve Document objects from the remote database. Defaults to True.

**Returns** Iterable of Document and/or DesignDocument objects

**admin_party**
Returns the CouchDB Admin Party status. `True` if using Admin Party `False` otherwise.

**Returns** CouchDB Admin Party mode status

**all_docs**(***kwargs*)
Wraps the _all_docs primary index on the database, and returns the results by value. This can be used as a direct query to the _all_docs endpoint. More convenient/efficient access using keys, slicing and iteration can be done through the `result` attribute.

Keyword arguments supported are those of the view/index access API.

**Parameters**

- **descending** (*bool*) – Return documents in descending key order.
- **endkey** – Stop returning records at this specified key.
- **endkey_docid** (*str*) – Stop returning records when the specified document id is reached.
- **include_docs** (*bool*) – Include the full content of the documents.
- **inclusive_end** (*bool*) – Include rows with the specified endkey.
- **key** – Return only documents that match the specified key.
- **keys** (*list*) – Return only documents that match the specified keys.
- **limit** (*int*) – Limit the number of returned documents to the specified count.
- **skip** (*int*) – Skip this number of rows from the start.
- **startkey** – Return records starting with the specified key.
- **startkey_docid** (*str*) – Return records starting with the specified document ID.

**Returns** Raw JSON response content from `_all_docs` endpoint

**bulk_docs**(*docs*)

Performs multiple document inserts and/or updates through a single request. Each document must either be or extend a dict as is the case with Document and DesignDocument objects. A document must contain the `_id` and `_rev` fields if the document is meant to be updated.

**Parameters** **docs** (*list*) – List of Documents to be created/updated.

**Returns** Bulk document creation/update status in JSON format

**changes**(*raw_data=False*, ***kwargs*)

Returns the `_changes` feed iterator. The `_changes` feed can be iterated over and once complete can also provide the last sequence identifier of the feed. If necessary, the iteration can be stopped by issuing a call to the `stop()` method on the returned iterator object.

For example:

```
# Iterate over a "normal" _changes feed
changes = db.changes()
for change in changes:
    print(change)
print(changes.last_seq)

# Iterate over a "continuous" _changes feed with additional options
changes = db.changes(feed='continuous', since='now', descending=True)
for change in changes:
    if some_condition:
        changes.stop()
    print(change)
```

**Parameters**

- **raw_data** (*bool*) – If set to True then the raw response data will be streamed otherwise if set to False then JSON formatted data will be streamed. Default is False.
- **conflicts** (*bool*) – Can only be set if include_docs is True. Adds information about conflicts to each document. Default is False.
- **descending** (*bool*) – Changes appear in sequential order. Default is False.

- **doc_ids** (*list*) – To be used only when `filter` is set to `_doc_ids`. Filters the feed so that only changes to the specified documents are sent.
- **feed** (*str*) – Type of feed. Valid values are `continuous`, `longpoll`, and `normal`. Default is `normal`.
- **filter** (*str*) – Name of filter function from a design document to get updates. Default is no filter.
- **heartbeat** (*int*) – Time in milliseconds after which an empty line is sent during `longpoll` or `continuous` if there have been no changes. Must be a positive number. Default is no heartbeat.
- **include_docs** (*bool*) – Include the document with the result. The document will not be returned as a *Document* but instead will be returned as either formated JSON or as raw response content. Default is False.
- **limit** (*int*) – Maximum number of rows to return. Must be a positive number. Default is no limit.
- **since** – Start the results from changes after the specified sequence identifier. In other words, using since excludes from the list all changes up to and including the specified sequence identifier. If since is 0 (the default), or omitted, the request returns all changes. If it is `now`, only changes made after the time of the request will be emitted.
- **style** (*str*) – Specifies how many revisions are returned in the changes array. The default, `main_only`, only returns the current "winning" revision; `all_docs` returns all leaf revisions, including conflicts and deleted former conflicts.
- **timeout** (*int*) – Number of milliseconds to wait for data before terminating the response. `heartbeat` supersedes `timeout` if both are supplied.
- **chunk_size** (*int*) – The HTTP response stream chunk size. Defaults to 512.

**Returns** Feed object that can be iterated over as a `_changes` feed.

**create**(*throw_on_exists=False*)

Creates a database defined by the current database object, if it does not already exist and raises a CloudantException if the operation fails. If the database already exists then this method call is a no-op.

**Parameters** **throw_on_exists** (*bool*) – Boolean flag dictating whether or not to throw a CloudantDatabaseException when attempting to create a database that already exists.

**Returns** The database object

**create_document**(*data*, *throw_on_exists=False*)

Creates a new document in the remote and locally cached database, using the data provided. If an _id is included in the data then depending on that _id either a *Document* or a *DesignDocument* object will be added to the locally cached database and returned by this method.

**Parameters**

- **data** (*dict*) – Dictionary of document JSON data, containing _id.
- **throw_on_exists** (*bool*) – Optional flag dictating whether to raise an exception if the document already exists in the database.

**Returns** A *Document* or *DesignDocument* instance corresponding to the new document in the database.

**create_query_index**(*design_document_id=None*, *index_name=None*, *index_type='json'*, ***kwargs*)

Creates either a JSON or a text query index in the remote database.

**Parameters**

- **index_type** (*str*) – The type of the index to create. Can be either 'text' or 'json'. Defaults to 'json'.
- **design_document_id** (*str*) – Optional identifier of the design document in which the index will be created. If omitted the default is that 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.
- **index_name** (*str*) – Optional name of the index. If omitted, a name will be generated automatically.
- **fields** (*list*) – A list of fields that should be indexed. For JSON indexes, the fields parameter is mandatory and should follow the 'sort syntax'. For example `fields=['name', {'age': 'desc'}]` will create an index on the 'name' field in ascending order and the 'age' field in descending order. For text indexes, the fields parameter is optional. If it is included then each field element in the fields list must be a single element dictionary where the key is the field name and the value is the field type. For example `fields=[{'name': 'string'}, {'age': 'number'}]`. Valid field types are `'string'`, `'number'`, and `'boolean'`.
- **default_field** (*dict*) – Optional parameter that specifies how the `$text` operator can be used with the index. Only valid when creating a text index.
- **selector** (*dict*) – Optional parameter that can be used to limit the index to a specific set of documents that match a query. It uses the same syntax used for selectors in queries. Only valid when creating a text index.

**Returns** An Index object representing the index created in the remote database

**creds**

Retrieves a dictionary of useful authentication information that can be used to authenticate against this database.

**Returns** Dictionary containing authentication information

**custom_result**(***kwds*)

Provides a context manager that can be used to customize the `_all_docs` behavior and wrap the output as a *Result*.

**Parameters**

- **descending** (*bool*) – Return documents in descending key order.
- **endkey** – Stop returning records at this specified key. Not valid when used with *Result* key access and key slicing.
- **endkey_docid** (*str*) – Stop returning records when the specified document id is reached.
- **include_docs** (*bool*) – Include the full content of the documents.
- **inclusive_end** (*bool*) – Include rows with the specified endkey.
- **key** – Return only documents that match the specified key. Not valid when used with *Result* key access and key slicing.
- **keys** (*list*) – Return only documents that match the specified keys. Not valid when used with *Result* key access and key slicing.
- **page_size** (*int*) – Sets the page size for result iteration.

- **startkey** – Return records starting with the specified key. Not valid when used with *Result* key access and key slicing.
- **startkey_docid** (*str*) – Return records starting with the specified document ID.

For example:

```
with database.custom_result(include_docs=True) as rslt:
    data = rslt[100: 200]
```

**database_url**
Constructs and returns the database URL.

**Returns** Database URL

**delete**()
Deletes the current database from the remote instance.

**delete_query_index**(*design_document_id*, *index_type*, *index_name*)
Deletes the query index identified by the design document id, index type and index name from the remote database.

**Parameters**

- **design_document_id** (*str*) – The design document id that the index exists in.
- **index_type** (*str*) – The type of the index to be deleted. Must be either 'text' or 'json'.
- **index_name** (*str*) – The index name of the index to be deleted.

**design_documents**()
Retrieve the JSON content for all design documents in this database. Performs a remote call to retrieve the content.

**Returns** All design documents found in this database in JSON format

**doc_count**()
Retrieves the number of documents in the remote database

**Returns** Database document count

**exists**()
Performs an existence check on the remote database.

**Returns** Boolean True if the database exists, False otherwise

**get_design_document**(*ddoc_id*)
Retrieves a design document. If a design document exists remotely then that content is wrapped in a DesignDocument object and returned to the caller. Otherwise a "shell" DesignDocument object is returned.

**Parameters** **ddoc_id** (*str*) – Design document id

**Returns** A DesignDocument instance, if exists remotely then it will be populated accordingly

**get_list_function_result**(*ddoc_id*, *list_name*, *view_name*, ***kwargs*)
Retrieves a customized MapReduce view result from the specified database based on the list function provided. List functions are used, for example, when you want to access Cloudant directly from a browser, and need data to be returned in a different format, such as HTML.

Note: All query parameters for View requests are supported. See `get_view_result` for all supported query parameters.

For example:

```
# Assuming that 'view001' exists as part of the
# 'ddoc001' design document in the remote database...
# Retrieve documents where the list function is 'list1'
resp = db.get_list_result('ddoc001', 'list1', 'view001', limit=10)
for row in resp['rows']:
    # Process data (in text format).
```

For more detail on list functions, refer to the Cloudant list documentation.

**Parameters**

- **ddoc_id** (*str*) – Design document id used to get result.
- **list_name** (*str*) – Name used in part to identify the list function.
- **view_name** (*str*) – Name used in part to identify the view.

**Returns** Formatted view result data in text format

**get_query_indexes**(*raw_result=False*)

Retrieves query indexes from the remote database.

**Parameters** **raw_result** (*bool*) – If set to True then the raw JSON content for the request is returned. Default is to return a list containing *Index*, *TextIndex*, and *SpecialIndex* wrapped objects.

**Returns** The query indexes in the database

**get_query_result**(*selector*, *fields=None*, *raw_result=False*, ***kwargs*)

Retrieves the query result from the specified database based on the query parameters provided. By default the result is returned as a *QueryResult* which uses the `skip` and `limit` query parameters internally to handle slicing and iteration through the query result collection. Therefore `skip` and `limit` cannot be used as arguments to get the query result when `raw_result=False`. However, by setting `raw_result=True`, the result will be returned as the raw JSON response content for the query requested. Using this setting requires the developer to manage their own slicing and iteration. Therefore `skip` and `limit` are valid arguments in this instance.

For example:

```
# Retrieve documents where the name field is 'foo'
selector = {'name': {'$eq': 'foo'}}
docs = db.get_query_result(selector)
for doc in docs:
    print doc

# Retrieve documents sorted by the age field in ascending order
docs = db.get_query_result(selector, sort=['name'])
for doc in docs:
    print doc

# Retrieve JSON response content, limiting response to 100 documents
resp = db.get_query_result(selector, raw_result=True, limit=100)
for doc in resp['docs']:
    print doc
```

For more detail on slicing and iteration, refer to the *QueryResult* documentation.

**Parameters**

- **selector** (*str*) – Dictionary object describing criteria used to select documents.
- **fields** (*list*) – A list of fields to be returned by the query.

- **raw_result** (*bool*) – Dictates whether the query result is returned wrapped in a QueryResult or if the response JSON is returned. Defaults to False.
- **bookmark** (*str*) – A string that enables you to specify which page of results you require. Only valid for queries using indexes of type *text*.
- **limit** (*int*) – Maximum number of results returned. Only valid if used with `raw_result=True`.
- **page_size** (*int*) – Sets the page size for result iteration. Default is 100. Only valid with `raw_result=False`.
- **r** (*int*) – Read quorum needed for the result. Each document is read from at least 'r' number of replicas before it is returned in the results.
- **skip** (*int*) – Skip the first 'n' results, where 'n' is the value specified. Only valid if used with `raw_result=True`.
- **sort** (*list*) – A list of fields to sort by. Optionally the list can contain elements that are single member dictionary structures that specify sort direction. For example `sort=['name', {'age': 'desc'}]` means to sort the query results by the "name" field in ascending order and the "age" field in descending order.
- **use_index** (*str*) – Identifies a specific index for the query to run against, rather than using the Cloudant Query algorithm which finds what it believes to be the best index.

**Returns** The result content either wrapped in a QueryResult or as the raw response JSON content

**get_revision_limit**()

Retrieves the limit of historical revisions to store for any single document in the current remote database.

**Returns** Revision limit value for the current remote database

**get_security_document**()

Retrieves the database security document as a SecurityDocument object. The returned object is useful for viewing as well as updating the the database's security document.

**Returns** A SecurityDocument instance representing the database security document

**get_show_function_result**(*ddoc_id*, *show_name*, *doc_id*)

Retrieves a formatted document from the specified database based on the show function provided. Show functions, for example, are used when you want to access Cloudant directly from a browser, and need data to be returned in a different format, such as HTML.

For example:

```
# Assuming that 'view001' exists as part of the
# 'ddoc001' design document in the remote database...
# Retrieve a formatted 'doc001' document where the show function is 'show001'
resp = db.get_show_function_result('ddoc001', 'show001', 'doc001')
for row in resp['rows']:
    # Process data (in text format).
```

For more detail on show functions, refer to the Cloudant show documentation.

**Parameters**

- **ddoc_id** (*str*) – Design document id used to get the result.
- **show_name** (*str*) – Name used in part to identify the show function.
- **doc_id** (*str*) – The ID of the document to show.

**Returns** Formatted document result data in text format

**get_view_result**(*ddoc_id*, *view_name*, *raw_result=False*, ***kwargs*)

Retrieves the view result based on the design document and view name. By default the result is returned as a *Result* object which provides a key accessible, sliceable, and iterable interface to the result collection. Depending on how you are accessing, slicing or iterating through your result collection certain query parameters are not permitted. See *Result* for additional details.

However, by setting `raw_result=True`, the result will be returned as the raw JSON response content for the view requested. With this setting there are no restrictions on the query parameters used but it also means that the result collection key access, slicing, and iteration is the responsibility of the developer.

For example:

```
# get Result based on a design document view
result = db.get_view_result('_design/ddoc_id_001', 'view_001')

# get a customized Result based on a design document view
result = db.get_view_result('_design/ddoc_id_001', 'view_001',
    include_docs=True, reduce=False)

# get raw response content based on a design document view
result = db.get_view_result('_design/ddoc_id_001', 'view_001',
    raw_result=True)

# get customized raw response content for a design document view
db.get_view_result('_design/ddoc_id_001', 'view_001',
    raw_result=True, include_docs=True, skip=100, limit=100)
```

For more detail on key access, slicing and iteration, refer to the *Result* documentation.

**Parameters**

- **ddoc_id** (*str*) – Design document id used to get result.
- **view_name** (*str*) – Name of the view used to get result.
- **raw_result** (*bool*) – Dictates whether the view result is returned as a default Result object or a raw JSON response. Defaults to False.
- **descending** (*bool*) – Return documents in descending key order.
- **endkey** – Stop returning records at this specified key. Not valid when used with *Result* key access and key slicing.
- **endkey_docid** (*str*) – Stop returning records when the specified document id is reached.
- **group** (*bool*) – Using the reduce function, group the results to a group or single row.
- **group_level** – Only applicable if the view uses complex keys: keys that are lists. Groups reduce results for the specified number of list fields.
- **include_docs** (*bool*) – Include the full content of the documents.
- **inclusive_end** (*bool*) – Include rows with the specified endkey.
- **key** – Return only documents that match the specified key. Not valid when used with *Result* key access and key slicing.
- **keys** (*list*) – Return only documents that match the specified keys. Not valid when used with *Result* key access and key slicing.
- **limit** (*int*) – Limit the number of returned documents to the specified count. Not valid when used with *Result* iteration.

- **page_size** (*int*) – Sets the page size for result iteration. Only valid if used with `raw_result=False`.
- **reduce** (*bool*) – True to use the reduce function, false otherwise.
- **skip** (*int*) – Skip this number of rows from the start. Not valid when used with *Result* iteration.
- **stable** (*bool*) – Whether or not the view results should be returned from a "stable" set of shards.
- **stale** (*str*) – Allow the results from a stale view to be used. This makes the request return immediately, even if the view has not been completely built yet. If this parameter is not given, a response is returned only after the view has been built. Note that this parameter is deprecated and the appropriate combination of *stable* and *update* should be used instead.
- **startkey** – Return records starting with the specified key. Not valid when used with *Result* key access and key slicing.
- **startkey_docid** (*str*) – Return records starting with the specified document ID.
- **update** (*str*) – Determine whether the view in question should be updated prior to or after responding to the user. Valid values are: false: return results before updating the view; true: Return results after updating the view; lazy: Return the view results without waiting for an update, but update them immediately after the request.

**Returns** The result content either wrapped in a QueryResult or as the raw response JSON content

**infinite_changes**(***kwargs*)

Returns an infinite (perpetually refreshed) `_changes` feed iterator. If necessary, the iteration can be stopped by issuing a call to the `stop()` method on the returned iterator object.

For example:

```
# Iterate over an infinite _changes feed
changes = db.infinite_changes()
for change in changes:
    if some_condition:
        changes.stop()
    print(change)
```

**Parameters**

- **conflicts** (*bool*) – Can only be set if include_docs is True. Adds information about conflicts to each document. Default is False.
- **descending** (*bool*) – Changes appear in sequential order. Default is False.
- **doc_ids** (*list*) – To be used only when `filter` is set to `_doc_ids`. Filters the feed so that only changes to the specified documents are sent.
- **filter** (*str*) – Name of filter function from a design document to get updates. Default is no filter.
- **heartbeat** (*int*) – Time in milliseconds after which an empty line is sent if there have been no changes. Must be a positive number. Default is no heartbeat.
- **include_docs** (*bool*) – Include the document with the result. The document will not be returned as a *Document* but instead will be returned as either formated JSON or as raw response content. Default is False.
- **since** – Start the results from changes after the specified sequence identifier. In other words, using since excludes from the list all changes up to and including the specified

sequence identifier. If since is 0 (the default), or omitted, the request returns all changes. If it is `now`, only changes made after the time of the request will be emitted.

- **style** (*str*) – Specifies how many revisions are returned in the changes array. The default, `main_only`, only returns the current "winning" revision; `all_docs` returns all leaf revisions, including conflicts and deleted former conflicts.
- **timeout** (*int*) – Number of milliseconds to wait for data before terminating the response. `heartbeat` supersedes `timeout` if both are supplied.
- **chunk_size** (*int*) – The HTTP response stream chunk size. Defaults to 512.

**Returns** Feed object that can be iterated over as a `_changes` feed.

**keys**(*remote=False*)

Retrieves the list of document ids in the database. Default is to return only the locally cached document ids, specify remote=True to make a remote request to include all document ids from the remote database instance.

**Parameters** **remote** (*bool*) – Dictates whether the list of locally cached document ids are returned or a remote request is made to include an up to date list of document ids from the server. Defaults to False.

**Returns** List of document ids

**list_design_documents**()

Retrieves a list of design document names in this database. Performs a remote call to retrieve the content.

**Returns** List of names for all design documents in this database

**metadata**()

Retrieves the remote database metadata dictionary.

**Returns** Dictionary containing database metadata details

**missing_revisions**(*doc_id*, **revisions*)

Returns a list of document revision values that do not exist in the current remote database for the specified document id and specified list of revision values.

**Parameters**

- **doc_id** (*str*) – Document id to check for missing revisions against.
- **revisions** (*list*) – List of document revisions values to check against.

**Returns** List of missing document revision values

**new_document**()

Creates a new, empty document in the remote and locally cached database, auto-generating the _id.

**Returns** Document instance corresponding to the new document in the database

**r_session**

Returns the `r_session` from the client instance used by the database.

**Returns** Client `r_session`

**revisions_diff**(*doc_id*, **revisions*)

Returns the differences in the current remote database for the specified document id and specified list of revision values.

**Parameters**

- **doc_id** (*str*) – Document id to check for revision differences against.

- **revisions** (*list*) – List of document revisions values to check against.

**Returns** The revision differences in JSON format

**set_revision_limit**(*limit*)
Sets the limit of historical revisions to store for any single document in the current remote database.

**Parameters** **limit** (*int*) – Number of revisions to store for any single document in the current remote database.

**Returns** Revision limit set operation status in JSON format

**update_handler_result**(*ddoc_id*, *handler_name*, *doc_id=None*, *data=None*, ***params*)
Creates or updates a document from the specified database based on the update handler function provided. Update handlers are used, for example, to provide server-side modification timestamps, and document updates to individual fields without the latest revision. You can provide query parameters needed by the update handler function using the `params` argument.

Create a document with a generated ID:

```
# Assuming that 'update001' update handler exists as part of the
# 'ddoc001' design document in the remote database...
# Execute 'update001' to create a new document
resp = db.update_handler_result('ddoc001', 'update001', data={'name': 'John',
                                'message': 'hello'})
```

Create or update a document with the specified ID:

```
# Assuming that 'update001' update handler exists as part of the
# 'ddoc001' design document in the remote database...
# Execute 'update001' to update document 'doc001' in the database
resp = db.update_handler_result('ddoc001', 'update001', 'doc001',
                                data={'month': 'July'})
```

For more details, see the update handlers documentation.

**Parameters**

- **ddoc_id** (*str*) – Design document id used to get result.
- **handler_name** (*str*) – Name used in part to identify the update handler function.
- **doc_id** (*str*) – Optional document id used to specify the document to be handled.

**Returns** Result of update handler function in text format

**view_cleanup**()
Removes view files that are not used by any design document in the remote database.

**Returns** View cleanup status in JSON format

### 3.1.3 document

API module/class for interacting with a document in a database.

**class** cloudant.document.**Document**(*database*, *document_id=None*, ***kwargs*)
Bases: dict

Encapsulates a JSON document. A Document object is instantiated with a reference to a database and used to manipulate document content in a CouchDB or Cloudant database instance.

In addition to basic CRUD style operations, a Document object also provides a convenient context manager. This context manager removes having to explicitly *fetch()* the document from the remote database before commencing work on it as well as explicitly having to *save()* the document once work is complete.

For example:

```
# Upon entry into the document context, fetches the document from the
# remote database, if it exists. Upon exit from the context, saves the
# document to the remote database with changes made within the context.
with Document(database, 'julia006') as document:
    # The document is fetched from the remote database
    # Changes are made locally
    document['name'] = 'Julia'
    document['age'] = 6
    # The document is saved to the remote database
```

**Parameters**

- **database** – A database instance used by the Document. Can be either a `CouchDatabase` or `CloudantDatabase` instance.
- **document_id** (*str*) – Optional document id used to identify the document.
- **encoder** (*str*) – Optional JSON encoder object (extending json.JSONEncoder).
- **decoder** (*str*) – Optional JSON decoder object (extending json.JSONDecoder).

**create**()
Creates the current document in the remote database and if successful, updates the locally cached Document object with the `_id` and `_rev` returned as part of the successful response.

**delete**()
Removes the document from the remote database and clears the content of the locally cached Document object with the exception of the `_id` field. In order to successfully remove a document from the remote database, a `_rev` value must exist in the locally cached Document object.

**delete_attachment**(*attachment*, *headers=None*)
Removes an attachment from a remote document and refreshes the locally cached document object.

**Parameters**

- **attachment** (*str*) – Attachment file name used to identify the attachment.
- **headers** (*dict*) – Optional, additional headers to be sent with request.

**Returns** Attachment deletion status in JSON format

**document_url**
Constructs and returns the document URL.

**Returns** Document URL

**exists**()
Retrieves whether the document exists in the remote database or not.

**Returns** True if the document exists in the remote database, otherwise False

**fetch**()
Retrieves the content of the current document from the remote database and populates the locally cached Document object with that content. A call to fetch will overwrite any dictionary content currently in the locally cached Document object.

**static field_set**(*doc*, *field*, *value*)
Sets or replaces a value for a field in a locally cached Document object. To remove the field set the `value` to None.

**Parameters**

- **doc** (*Document*) – Locally cached Document object that can be a Document, DesignDocument or dict.
- **field** (*str*) – Name of the field to set.
- **value** – Value to set the field to.

**get_attachment**(*attachment*, *headers=None*, *write_to=None*, *attachment_type=None*)
Retrieves a document's attachment and optionally writes it to a file. If the content_type of the attachment is 'application/json' then the data returned will be in JSON format otherwise the response content will be returned as text or binary.

**Parameters**

- **attachment** (*str*) – Attachment file name used to identify the attachment.
- **headers** (*dict*) – Optional, additional headers to be sent with request.
- **write_to** (*file*) – Optional file handler to write the attachment to. The write_to file must be opened for writing prior to including it as an argument for this method.
- **attachment_type** (*str*) – Optional setting to define how to handle the attachment when returning its contents from this method. Valid values are `'text'`, `'json'`, and `'binary'` If omitted then the returned content will be based on the response Content-Type.

**Returns** The attachment content

**json**()
Retrieves the JSON string representation of the current locally cached document object, encoded by the encoder specified in the associated client object.

**Returns** Encoded JSON string containing the document data

**static list_field_append**(*doc*, *field*, *value*)
Appends a value to a list field in a locally cached Document object. If a field does not exist it will be created first.

**Parameters**

- **doc** (*Document*) – Locally cached Document object that can be a Document, DesignDocument or dict.
- **field** (*str*) – Name of the field list to append to.
- **value** – Value to append to the field list.

**static list_field_remove**(*doc*, *field*, *value*)
Removes a value from a list field in a locally cached Document object.

**Parameters**

- **doc** (*Document*) – Locally cached Document object that can be a Document, DesignDocument or dict.
- **field** (*str*) – Name of the field list to remove from.
- **value** – Value to remove from the field list.

**put_attachment**(*attachment*, *content_type*, *data*, *headers=None*)
Adds a new attachment, or updates an existing attachment, to the remote document and refreshes the locally cached Document object accordingly.

**Parameters**

- **attachment** – Attachment file name used to identify the attachment.
- **content_type** – The http `Content-Type` of the attachment used as an additional header.
- **data** – Attachment data defining the attachment content.
- **headers** – Optional, additional headers to be sent with request.

**Returns** Attachment addition/update status in JSON format

**r_session**
Returns the database instance `r_session` used by the document.

**Returns** Client `r_session`

**save**()
Saves changes made to the locally cached Document object's data structures to the remote database. If the document does not exist remotely then it is created in the remote database. If the object does exist remotely then the document is updated remotely. In either case the locally cached Document object is also updated accordingly based on the successful response of the operation.

**update_field**(*action*, *field*, *value*, *max_tries=10*)
Updates a field in the remote document. If a conflict exists, the document is re-fetched from the remote database and the update is retried. This is performed up to `max_tries` number of times.

Use this method when you want to update a single field in a document, and don't want to risk clobbering other people's changes to the document in other fields, but also don't want the caller to implement logic to deal with conflicts.

For example:

```
# Append the string 'foo' to the 'words' list of Document doc.
doc.update_field(
    action=doc.list_field_append,
    field='words',
    value='foo'
)
```

**Parameters**

- **action** (*callable*) – A routine that takes a Document object, a field name, and a value. The routine should attempt to update a field in the locally cached Document object with the given value, using whatever logic is appropriate. Valid actions are *list_field_append()*, *list_field_remove()*, *field_set()*
- **field** (*str*) – Name of the field to update
- **value** – Value to update the field with
- **max_tries** (*int*) – In the case of a conflict, the number of retries to attempt

### 3.1.4 design_document

API module/class for interacting with a design document in a database.

**class** cloudant.design_document.**DesignDocument**(*database*, *document_id=None*)

Bases: *cloudant.document.Document*

Encapsulates a specialized version of a *Document*. A DesignDocument object is instantiated with a reference to a database and provides an API to view management, index management, list and show functions, etc. When instantiating a DesignDocument or when setting the document id (_id) field, the value must start with _design/. If it does not, then _design/ will be prepended to the provided document id value.

Note: Currently only the view management and search index management API exists. Remaining design document functionality will be added later.

**Parameters**

- **database** – A database instance used by the DesignDocument. Can be either a CouchDatabase or CloudantDatabase instance.
- **document_id** (*str*) – Optional document id. If provided and does not start with _design/, it will be prepended with _design/.

**add_list_function**(*list_name*, *list_func*)

Appends a list function to the locally cached DesignDocument indexes dictionary.

**Parameters**

- **list_name** (*str*) – Name used to identify the list function.
- **list_func** (*str*) – Javascript list function.

**add_search_index**(*index_name*, *search_func*, *analyzer=None*)

Appends a Cloudant search index to the locally cached DesignDocument indexes dictionary.

**Parameters**

- **index_name** (*str*) – Name used to identify the search index.
- **search_func** (*str*) – Javascript search index function.
- **analyzer** – Optional analyzer for this search index.

**add_show_function**(*show_name*, *show_func*)

Appends a show function to the locally cached DesignDocument shows dictionary.

**Parameters**

- **show_name** – Name used to identify the show function.
- **show_func** – Javascript show function.

**add_view**(*view_name*, *map_func*, *reduce_func=None*, ***kwargs*)

Appends a MapReduce view to the locally cached DesignDocument View dictionary. To create a JSON query index use create_query_index() instead. A CloudantException is raised if an attempt to add a QueryIndexView (JSON query index) using this method is made.

**Parameters**

- **view_name** (*str*) – Name used to identify the View.
- **map_func** (*str*) – Javascript map function.
- **reduce_func** (*str*) – Optional Javascript reduce function.

**delete_index**(*index_name*)

Removes an existing index in the locally cached DesignDocument indexes dictionary.

**Parameters** **index_name** (*str*) – Name used to identify the index.

**delete_list_function**(*list_name*)
Removes an existing list function in the locally cached DesignDocument lists dictionary.

**Parameters** **list_name** (*str*) – Name used to identify the list.

**delete_show_function**(*show_name*)
Removes an existing show function in the locally cached DesignDocument shows dictionary.

**Parameters** **show_name** – Name used to identify the list.

**delete_view**(*view_name*)
Removes an existing MapReduce view definition from the locally cached DesignDocument View dictionary. To delete a JSON query index use `delete_query_index()` instead. A CloudantException is raised if an attempt to delete a QueryIndexView (JSON query index) using this method is made.

**Parameters** **view_name** (*str*) – Name used to identify the View.

**fetch**()
Retrieves the remote design document content and populates the locally cached DesignDocument dictionary. View content is stored either as View or QueryIndexView objects which are extensions of the `dict` type. All other design document data are stored directly as `dict` types.

**filters**
Provides an accessor property to the filters dictionary in the locally cached DesignDocument. Filter functions enable you to add tests for filtering each of the objects included in the changes feed. If any of the function tests fail, the object is filtered from the feed. If the function returns a true result when applied to a change, the change remains in the feed.

Filter functions require two arguments: `doc` and `req`. The `doc` argument represents the document being tested for filtering. The `req` argument contains additional information about the HTTP request.

Filter function example:

```
# Add the filter function to ``filters`` and save the design document
ddoc = DesignDocument(self.db, '_design/ddoc001')
# Filter and remove documents that are not of ``type`` mail
ddoc['filters'] = {
    'filter001': 'function(doc, req){if (doc.type != 'mail'){return false;} '
                 'return true;} '
}
ddoc.save()
```

To execute filter functions on a changes feed, see the database API *changes()*

For more details, see the Filter functions documentation.

**Returns** Dictionary containing filter function names and functions as key/value

**get_index**(*index_name*)
Retrieves a specific index from the locally cached DesignDocument indexes dictionary by name.

**Parameters** **index_name** (*str*) – Name used to identify the index.

**Returns** Index dictionary for the specified index name

**get_list_function**(*list_name*)
Retrieves a specific list function from the locally cached DesignDocument lists dictionary by name.

**Parameters** **list_name** (*str*) – Name used to identify the list function.

**Returns** String form of the specified list function

**get_show_function**(*show_name*)
Retrieves a specific show function from the locally cached DesignDocument shows dictionary by name.

**Parameters** **show_name** (*str*) – Name used to identify the show function.

**Returns** String form of the specified show function

**get_view**(*view_name*)
Retrieves a specific View from the locally cached DesignDocument by name.

**Parameters** **view_name** (*str*) – Name used to identify the View.

**Returns** View object for the specified view_name

**indexes**
Provides an accessor property to the indexes dictionary in the locally cached DesignDocument.

**Returns** Dictionary containing index names and index objects as key/value

**iterindexes**()
Provides a way to iterate over the locally cached DesignDocument indexes dictionary.

For example:

```
for index_name, search_func in ddoc.iterindexes():
    # Perform search index processing
```

**Returns** Iterable containing index name and associated index object

**iterlists**()
Provides a way to iterate over the locally cached DesignDocument lists dictionary.

**Returns** Iterable containing list function name and associated list function

**itershows**()
Provides a way to iterate over the locally cached DesignDocument shows dictionary.

**Returns** Iterable containing show function name and associated show function

**iterviews**()
Provides a way to iterate over the locally cached DesignDocument View dictionary.

For example:

```
for view_name, view in ddoc.iterviews():
    # Perform view processing
```

**Returns** Iterable containing view name and associated View object

**list_indexes**()
Retrieves a list of available indexes in the locally cached DesignDocument.

**Returns** List of index names

**list_list_functions**()
Retrieves a list of available list functions in the locally cached DesignDocument lists dictionary.

**Returns** List of list function names

**list_show_functions**()
Retrieves a list of available show functions in the locally cached DesignDocument shows dictionary.

**Returns** List of show function names

**list_views**()
Retrieves a list of available View objects in the locally cached DesignDocument.

**Returns** List of view names

**lists**
Provides an accessor property to the lists dictionary in the locally cached DesignDocument.

**Returns** Dictionary containing list names and objects as key/value

**rewrites**
Provides an accessor property to a list of dictionaries with rewrite rules in the locally cached DesignDocument. Each rule for URL rewriting is a JSON object with four fields: `from`, `to`, `method`, and `query`.

Note: Requests that match the rewrite rules must have a URL path that starts with `/$DATABASE/_design/doc/_rewrite`.

Rewrite rule example:

```
# Add the rule to ``rewrites`` and save the design document
ddoc = DesignDocument(self.db, '_design/ddoc001')
ddoc['rewrites'] = [
    {"from": "/old/topic",
     "to": "/new/",
     "method": "GET",
     "query": {}
     }
]
ddoc.save()
```

Once the rewrite rule is saved to the remote database, the GET request URL `/$DATABASE/_design/doc/_rewrite/old/topic?k=v` would be rewritten as `/$DATABASE/_design/doc/_rewrite/new?k=v`.

For more details on URL rewriting, see the rewrite rules documentation.

**Returns** List of dictionaries containing rewrite rules as key/value

**save**()
Saves changes made to the locally cached DesignDocument object's data structures to the remote database. If the design document does not exist remotely then it is created in the remote database. If the object does exist remotely then the design document is updated remotely. In either case the locally cached DesignDocument object is also updated accordingly based on the successful response of the operation.

**search_disk_size**(*search_index*)
Retrieves disk size information about a specified search index within the design document, returns dictionary

GET databasename/_design/{ddoc}/_search_disk_size/{search_index}

**search_info**(*search_index*)
Retrieves information about a specified search index within the design document, returns dictionary

GET databasename/_design/{ddoc}/_search_info/{search_index}

**shows**
Provides an accessor property to the shows dictionary in the locally cached DesignDocument.

**Returns** Dictionary containing show names and functions as key/value

**st_indexes**
Provides an accessor property to the Cloudant Geospatial (a.k.a. Cloudant Geo) indexes dictionary in the locally cached DesignDocument. Each Cloudant Geo index is a JSON object within the `st_indexes` containing an index name and a javascript function.

Note: To make it easier to work with Cloudant Geo documents, it is best practice to create a separate design document specifically for Cloudant Geo indexes.

Geospatial index example:

```
# Add the Cloudant Geo index to ``st_indexes`` and save the design document
ddoc = DesignDocument(self.db, '_design/ddoc001')
ddoc['st_indexes'] = {
    'geoidx': {
        'index': 'function(doc) { '
                 'if (doc.geometry && doc.geometry.coordinates) { '
                 'st_index(doc.geometry);}} '
    }
}
ddoc.save()
```

Once the Cloudant Geo index is saved to the remote database, you can query the index with a GET request. To issue a request against the `_geo` endpoint, see the steps outlined in the endpoint access section.

For more details, see the Cloudant Geospatial documentation.

**Returns** Dictionary containing Cloudant Geo names and index objects as key/value

**update_list_function**(*list_name*, *list_func*)

Modifies/overwrites an existing list function in the locally cached DesignDocument indexes dictionary.

**Parameters**

- **list_name** (*str*) – Name used to identify the list function.
- **list_func** (*str*) – Javascript list function.

**update_search_index**(*index_name*, *search_func*, *analyzer=None*)

Modifies/overwrites an existing Cloudant search index in the locally cached DesignDocument indexes dictionary.

**Parameters**

- **index_name** (*str*) – Name used to identify the search index.
- **search_func** (*str*) – Javascript search index function.
- **analyzer** – Optional analyzer for this search index.

**update_show_function**(*show_name*, *show_func*)

Modifies/overwrites an existing show function in the locally cached DesignDocument shows dictionary.

**Parameters**

- **show_name** – Name used to identify the show function.
- **show_func** – Javascript show function.

**update_view**(*view_name*, *map_func*, *reduce_func=None*, ***kwargs*)

Modifies/overwrites an existing MapReduce view definition in the locally cached DesignDocument View dictionary. To update a JSON query index use `delete_query_index()` followed by `create_query_index()` instead. A CloudantException is raised if an attempt to update a QueryIndexView (JSON query index) using this method is made.

**Parameters**

- **view_name** (*str*) – Name used to identify the View.
- **map_func** (*str*) – Javascript map function.

- **reduce_func** (*str*) – Optional Javascript reduce function.

**updates**

Provides an accessor property to the updates dictionary in the locally cached DesignDocument. Update handlers are custom functions stored on Cloudant's server that will create or update a document. To execute the update handler function, see *update_handler_result()*.

Update handlers receive two arguments: `doc` and `req`. If a document ID is provided in the request to the update handler, then `doc` will be the document corresponding with that ID. If no ID was provided, `doc` will be null.

Update handler example:

```
# Add the update handler to ``updates`` and save the design document
ddoc = DesignDocument(self.db, '_design/ddoc001')
ddoc001['updates'] = {
    'update001': 'function(doc, req) { if (!doc) '
                 '{ if ('id' in req && req.id){ return [{_id: req.id}, '
                 '"New World"] } return [null, "Empty World"] } '
                 'doc.world = 'hello'; '
                 'return [doc, "Added world.hello!"]} '
}
ddoc.save()
```

Note: Update handler functions must return an array of two elements, the first being the document to save (or null, if you don't want to save anything), and the second being the response body.

**Returns** Dictionary containing update handler names and objects as key/value

**validate_doc_update**

Provides an accessor property to the update validators dictionary in the locally cached DesignDocument. Update validators evaluate whether a document should be written to disk when insertions and updates are attempted.

Update validator example:

```
# Add the update validator to ``validate_doc_update`` and save the design
↪document
ddoc = DesignDocument(self.db, '_design/ddoc001')
ddoc['validate_doc_update'] = (
    'function(newDoc, oldDoc, userCtx, secObj) { '
    'if (newDoc.address === undefined) { '
    'throw({forbidden: 'Document must have an address.'}); }}')
ddoc.save()
```

For more details, see the Update Validators documentation.

**Returns** Dictionary containing update validator functions

**views**

Provides an accessor property to the View dictionary in the locally cached DesignDocument.

**Returns** Dictionary containing view names and View objects as key/value

### 3.1.5 security_document

API module/class for interacting with a security document in a database.

**class** cloudant.security_document.**SecurityDocument**(*database*)

Bases: dict

Encapsulates a JSON security document. A SecurityDocument object is instantiated with a reference to a database and used to manipulate security document content in a CouchDB or Cloudant database instance.

In addition to basic read/write operations, a SecurityDocument object also provides a convenient context manager. This context manager removes having to explicitly *fetch()* the security document from the remote database before commencing work on it as well as explicitly having to *save()* the security document once work is complete.

For example:

```
# Upon entry into the security document context, fetches the security
# document from the remote database, if it exists. Upon exit from the
# context, saves the security document to the remote database with
# changes made within the context.
with SecurityDocument(database) as security_document:
    # The security document is fetched from the remote database
    # Changes are made locally
    security_document['Cloudant']['julia'] = ['_reader', '_writer']
    security_document['Cloudant']['ruby'] = ['_admin', '_replicator']
    # The security document is saved to the remote database
```

**Parameters** **database** – A database instance used by the SecurityDocument. Can be either a `CouchDatabase` or `CloudantDatabase` instance.

**document_url**
Constructs and returns the security document URL.

**Returns** Security document URL

**fetch**()
Retrieves the content of the current security document from the remote database and populates the locally cached SecurityDocument object with that content. A call to fetch will overwrite any dictionary content currently in the locally cached SecurityDocument object.

**json**()
Retrieves the JSON string representation of the current locally cached security document object, encoded by the encoder specified in the associated client object.

**Returns** Encoded JSON string containing the security document data

**r_session**
Returns the Python requests session used by the security document.

**Returns** The Python requests session

**save**()
Saves changes made to the locally cached SecurityDocument object's data structures to the remote database.

### 3.1.6 view

API module for interacting with a view in a design document.

**class** cloudant.view.**QueryIndexView**(*ddoc*, *view_name*, *map_fields*, *reduce_func*, ***kwargs*)
Bases: *cloudant.view.View*

A view that defines a JSON query index in a design document.

If you wish to manage a view that represents a JSON query index it is strongly recommended that `create_query_index()` and `delete_query_index()` are used.

**__call__**(***kwargs*)
QueryIndexView objects are not callable. If you wish to execute a query using a query index, use `get_query_result()` instead.

**custom_result**(***options*)
This method overrides the View base class *custom_result()* method with the sole purpose of disabling it. Since QueryIndexView objects are not callable, there is no reason to wrap their output in a Result. If you wish to execute a query using a query index, use `get_query_result()` instead.

**map**
Provides a map property accessor and setter.

**Parameters** **map_func** (*dict*) – A dictionary of fields defining the index.

**Returns** Fields defining the index

**reduce**
Provides a reduce property accessor and setter.

**Parameters** **reduce_func** (*str*) – A string representation of the reduce function used in part to define the index.

**Returns** Reduce function as a string

**class** cloudant.view.**View**(*ddoc*, *view_name*, *map_func=None*, *reduce_func=None*, ***kwargs*)
Bases: dict

Encapsulates a view as a dictionary based object, exposing the map and reduce functions as attributes and supporting query/data access through the view. A View object is instantiated with a reference to a DesignDocument and is typically used as part of the *DesignDocument* view management API.

A View object provides a key accessible, sliceable, and iterable default result collection that can be used to query the view data through the `result` attribute.

For example:

```
# Access result collection through individual keys
view.result[100]
view.result['foo']

# Access result collection through index slicing:
view.result[100: 200]
view.result[: 200]
view.result[100: ]
view.result[: ]

# Access result collection through key slicing:
view.result['bar': 'foo']
view.result['bar': ]
view.result[: 'foo']

# Iterate over the result collection:
for doc in view.result:
    print doc
```

The default result collection provides basic functionality, which can be customized with other arguments using the *custom_result()* context manager.

For example:

```
# Including documents as part of a custom result
with view.custom_result(include_docs=True) as rslt:
    rslt[100: 200]                      # slice by result
    rslt[['2013', '10']: ['2013', '11']] # slice by startkey/endkey

    # Iteration
    for doc in rslt:
        print doc

# Iteration over a view within startkey/endkey range:
with view.custom_result(startkey='2013', endkey='2014') as rslt:
    for doc in rslt:
        print doc
```

Note: A view must exist as part of a design document remotely in order to access result content as depicted in the above examples.

**Parameters**

- **ddoc** (DesignDocument) – DesignDocument instance used in part to identify the view.
- **view_name** (*str*) – Name used in part to identify the view.
- **map_func** (*str*) – Optional Javascript map function.
- **reduce_func** (*str*) – Optional Javascript reduce function.

**__call__**(***kwargs*)

Makes the View object callable and retrieves the raw JSON content from the remote database based on the View definition on the server, using the kwargs provided as query parameters.

For example:

```
# Construct a View
view = View(ddoc, 'view001')
# Assuming that 'view001' exists as part of the
# design document ddoc in the remote database...
# Use view as a callable
for row in view(include_docs=True, limit=100, skip=100)['rows']:
    # Process view data (in JSON format).
```

Note: Rather than using the View callable directly, if you wish to retrieve view results in raw JSON format use `raw_result=True` with the provided database API of *`get_view_result()`* instead.

**Parameters**

- **descending** (*bool*) – Return documents in descending key order.
- **endkey** – Stop returning records at this specified key.
- **endkey_docid** (*str*) – Stop returning records when the specified document id is reached.
- **group** (*bool*) – Using the reduce function, group the results to a group or single row.
- **group_level** – Only applicable if the view uses complex keys: keys that are JSON arrays. Groups reduce results for the specified number of array fields.
- **include_docs** (*bool*) – Include the full content of the documents.
- **inclusive_end** (*bool*) – Include rows with the specified endkey.
- **key** (*str*) – Return only documents that match the specified key.

- **keys** (*list*) – Return only documents that match the specified keys.
- **limit** (*int*) – Limit the number of returned documents to the specified count.
- **reduce** (*bool*) – True to use the reduce function, false otherwise.
- **skip** (*int*) – Skip this number of rows from the start.
- **stale** (*str*) – Allow the results from a stale view to be used. This makes the request return immediately, even if the view has not been completely built yet. If this parameter is not given, a response is returned only after the view has been built.
- **startkey** – Return records starting with the specified key.
- **startkey_docid** (*str*) – Return records starting with the specified document ID.

**Returns** View result data in JSON format

**custom_result**(***kwds*)

Customizes the *Result* behavior and provides a convenient context manager for the Result. Result customizations can be made by providing extra options to the result call using this context manager. Depending on how you are accessing, slicing or iterating through your result collection certain query parameters are not permitted. See *Result* for additional details.

For example:

```
with view.custom_result(include_docs=True, reduce=False) as rslt:
    data = rslt[100: 200]
```

**Parameters**

- **descending** (*bool*) – Return documents in descending key order.
- **endkey** – Stop returning records at this specified key. Not valid when used with *Result* key access and key slicing.
- **endkey_docid** (*str*) – Stop returning records when the specified document id is reached.
- **group** (*bool*) – Using the reduce function, group the results to a group or single row.
- **group_level** – Only applicable if the view uses complex keys: keys that are JSON arrays. Groups reduce results for the specified number of array fields.
- **include_docs** (*bool*) – Include the full content of the documents.
- **inclusive_end** (*bool*) – Include rows with the specified endkey.
- **key** – Return only documents that match the specified key. Not valid when used with *Result* key access and key slicing.
- **keys** (*list*) – Return only documents that match the specified keys. Not valid when used with *Result* key access and key slicing.
- **limit** (*int*) – Limit the number of returned documents to the specified count. Not valid when used with *Result* iteration.
- **page_size** (*int*) – Sets the page size for result iteration.
- **reduce** (*bool*) – True to use the reduce function, false otherwise.
- **skip** (*int*) – Skip this number of rows from the start. Not valid when used with *Result* iteration.

- **stale** (*str*) – Allow the results from a stale view to be used. This makes the request return immediately, even if the view has not been completely built yet. If this parameter is not given, a response is returned only after the view has been built.
- **startkey** – Return records starting with the specified key. Not valid when used with *Result* key access and key slicing.
- **startkey_docid** (*str*) – Return records starting with the specified document ID.

**Returns** View result data wrapped in a Result instance

**map**

Provides an map property accessor and setter.

For example:

```
# Set the View map property
view.map = 'function (doc) {\n  emit(doc._id, 1);\n}'
print view.map
```

**Parameters** **js_func** (*str*) – Javascript function.

**Returns** Codified map function

**reduce**

Provides an reduce property accessor and setter.

For example:

```
# Set the View reduce property
view.reduce = '_count'
# Get and print the View reduce property
print view.reduce
```

**Parameters** **js_func** (*str*) – Javascript function.

**Returns** Codified reduce function

**url**

Constructs and returns the View URL.

**Returns** View URL

### 3.1.7 query

API module for composing and executing Cloudant queries.

**class** cloudant.query.**Query**(*database*, ***kwargs*)

Bases: dict

Encapsulates a query as a dictionary based object, providing a sliceable and iterable query result collection that can be used to process query output data through the `result` attribute.

For example:

```
# Slicing to skip/limit:
query.result[100:200]
query.result[:200]
```

(continues on next page)

(continued from previous page)

```
query.result[100:]
query.result[:]

# Iteration is supported via the result attribute:
for doc in query.result:
    print doc
```

The query `result` collection provides basic functionality, which can be customized with other arguments using the *custom_result()* context.

For example:

```
# Setting the read quorum as part of a custom result
with query.custom_result(r=3) as rslt:
    rslt[100:200] # slice the result

    # Iteration
    for doc in rslt:
        print doc

# Iteration over a query result sorted by the "name" field:
with query.custom_result(sort=[{'name': 'asc'}]) as rslt:
    for doc in rslt:
        print doc
```

**Parameters**

- **database** (*CloudantDatabase*) – A Cloudant database instance used by the Query.
- **bookmark** (*str*) – A string that enables you to specify which page of results you require. Only valid for queries using indexes of type *text*.
- **fields** (*list*) – A list of fields to be returned by the query.
- **limit** (*int*) – Maximum number of results returned.
- **r** (*int*) – Read quorum needed for the result. Each document is read from at least 'r' number of replicas before it is returned in the results.
- **selector** (*str*) – Dictionary object describing criteria used to select documents.
- **skip** (*int*) – Skip the first 'n' results, where 'n' is the value specified.
- **sort** (*list*) – A list of fields to sort by. Optionally the list can contain elements that are single member dictionary structures that specify sort direction. For example `sort=['name', {'age': 'desc'}]` means to sort the query results by the "name" field in ascending order and the "age" field in descending order.
- **use_index** (*str*) – Identifies a specific index for the query to run against, rather than using the Cloudant Query algorithm which finds what it believes to be the best index.

**__call__**(***kwargs*)

Makes the Query object callable and retrieves the raw JSON content from the remote database based on the current Query definition, and any additional kwargs provided as query parameters.

For example:

```
# Construct a Query
query = Query(database, selector={'_id': {'$gt': 0}})
```

(continues on next page)

(continued from previous page)

```
# Use query as a callable limiting results to 100,
# skipping the first 100.
for doc in query(limit=100, skip=100)['docs']:
    # Process query data (in JSON format).
```

Note: Rather than using the Query callable directly, if you wish to retrieve query results in raw JSON format use the provided database API of *get_query_result()* and set `raw_result=True` instead.

**Parameters**

- **bookmark** (*str*) – A string that enables you to specify which page of results you require. Only valid for queries using indexes of type *text*.
- **fields** (*list*) – A list of fields to be returned by the query.
- **limit** (*int*) – Maximum number of results returned.
- **r** (*int*) – Read quorum needed for the result. Each document is read from at least 'r' number of replicas before it is returned in the results.
- **selector** (*str*) – Dictionary object describing criteria used to select documents.
- **skip** (*int*) – Skip the first 'n' results, where 'n' is the value specified.
- **sort** (*list*) – A list of fields to sort by. Optionally the list can contain elements that are single member dictionary structures that specify sort direction. For example `sort=['name', {'age': 'desc'}]` means to sort the query results by the "name" field in ascending order and the "age" field in descending order.
- **use_index** (*str*) – Identifies a specific index for the query to run against, rather than using the Cloudant Query algorithm which finds what it believes to be the best index.

**Returns** Query result data in JSON format

**custom_result**(***kwds*)

Customizes the *QueryResult* behavior and provides a convenient context manager for the QueryResult. QueryResult customizations can be made by providing extra options to the query result call using this context manager. The use of `skip` and `limit` as options are not valid when using a QueryResult since the `skip` and `limit` functionality is handled in the QueryResult.

For example:

```
with query.custom_result(sort=[{'name': 'asc'}]) as rslt:
    data = rslt[100:200]
```

**Parameters**

- **bookmark** (*str*) – A string that enables you to specify which page of results you require. Only valid for queries using indexes of type *text*.
- **fields** (*list*) – A list of fields to be returned by the query.
- **page_size** (*int*) – Sets the page size for result iteration. Default is 100.
- **r** (*int*) – Read quorum needed for the result. Each document is read from at least 'r' number of replicas before it is returned in the results.
- **selector** (*str*) – Dictionary object describing criteria used to select documents.

- **sort** (*list*) – A list of fields to sort by. Optionally the list can contain elements that are single member dictionary structures that specify sort direction. For example `sort=['name', {'age': 'desc'}]` means to sort the query results by the "name" field in ascending order and the "age" field in descending order.
- **use_index** (*str*) – Identifies a specific index for the query to run against, rather than using the Cloudant Query algorithm which finds what it believes to be the best index.

**Returns** Query result data wrapped in a QueryResult instance

**url**
Constructs and returns the Query URL.

**Returns** Query URL

### 3.1.8 index

API module for managing/viewing query indexes.

**class** cloudant.index.**Index**(*database*, *design_document_id=None*, *name=None*, ***kwargs*)
Bases: object

Provides an interface for managing a JSON query index. Primarily meant to be used by the database convenience methods `create_query_index()`, `delete_query_index()`, and `get_query_indexes()`. It is recommended that you use those methods to manage an index rather than directly interfacing with Index objects.

**Parameters**

- **database** (CloudantDatabase) – A Cloudant database instance used by the Index.
- **design_document_id** (*str*) – Optional identifier of the design document.
- **name** (*str*) – Optional name of the index.
- **kwargs** – Options used to construct the index definition for the purposes of index creation. For more details on valid options See `create_query_index()`.

**as_a_dict**()
Displays the index as a dictionary. This includes the design document id, index name, index type, and index definition.

**Returns** Dictionary representation of the index as a dictionary

**create**()
Creates the current index in the remote database.

**definition**
Displays the index definition. This could be either the definiton to be used to construct the index or the definition as it is returned by a GET request to the *_index* endpoint.

**Returns** Index definition as a dictionary

**delete**()
Removes the current index from the remote database.

**design_document_id**
Displays the design document id.

**Returns** Design document that this index belongs to

**index_url**
Constructs and returns the index URL.

**Returns** Index URL

**name**
Displays the index name.

**Returns** Name for this index

**type**
Displays the index type.

**Returns** Type of this index

**class** cloudant.index.**SpecialIndex**(*database*, *design_document_id=None*, *name='_all_docs'*, ***kwargs*)
Bases: *cloudant.index.Index*

Provides an interface for viewing the "special" primary index of a database. Primarily meant to be used by the database convenience method `get_query_indexes()`. It is recommended that you use that method to view the "special" index rather than directly interfacing with the SpecialIndex object.

**create**()
A "special" index cannot be created. This method is disabled for a SpecialIndex object.

**delete**()
A "special" index cannot be deleted. This method is disabled for a SpecialIndex object.

**class** cloudant.index.**TextIndex**(*database*, *design_document_id=None*, *name=None*, ***kwargs*)
Bases: *cloudant.index.Index*

Provides an interface for managing a text query index. Primarily meant to be used by the database convenience methods `create_query_index()`, `delete_query_index()`, and `get_query_indexes()`. It is recommended that you use those methods to manage an index rather than directly interfacing with TextIndex objects.

**Parameters**

- **database** (CloudantDatabase) – A Cloudant database instance used by the TextIndex.
- **design_document_id** (*str*) – Optional identifier of the design document.
- **name** (*str*) – Optional name of the index.
- **kwargs** – Options used to construct the index definition for the purposes of index creation. For more details on valid options See `create_query_index()`.

### 3.1.9 result

API module for interacting with result collections.

**class** cloudant.result.**QueryResult**(*query*, ***options*)
Bases: *cloudant.result.Result*

Provides a index key accessible, sliceable and iterable interface to query result collections by extending the *Result* class. A QueryResult object is constructed with a raw data callable reference to the Query *__call__()* callable, which is used to retrieve data. A QueryResult object can also use optional extra arguments for result customization and supports efficient, paged iteration over the result collection to avoid large result data from adversely affecting memory.

In Python, slicing returns by value, whereas iteration will yield elements of the sequence. This means that index key access and slicing will perform better for smaller data collections, whereas iteration will be more efficient for larger data collections.

For example:

```
# Key access:

# Access by index value:
query_result = QueryResult(query)
query_result[9]         # skip first 9 documents and get 10th

# Slice access:

# Access by index slices:
query_result = QueryResult(query)
query_result[100: 200] # get documents after the 100th and up to and including␣
↪the 200th
query_result[ :200]    # get documents up to and including the 200th
query_result[100: ]    # get all documents after the 100th
query_result[: ]       # get all documents

# Iteration:

# Iterate over the entire result collection
query_result = QueryResult(query)
for doc in query_result:
    print doc

# Iterate over the result collection, with an overriding query sort
query_result = QueryResult(query, sort=[{'name': 'desc'}])
for doc in query_result:
    print doc

# Iterate over the entire result collection,
# explicitly setting the index and in batches of 1000.
query_result = QueryResult(query, use_index='my_index', page_size=1000)
for doc in query_result:
    print doc
```

Note: Only access by index value, slicing by index values and iteration are supported by QueryResult. Also, since QueryResult object iteration uses the `skip` and `limit` query parameters to handle its processing, `skip` and `limit` are not permitted to be part of the query callable or be included as part of the QueryResult customized parameters.

**Parameters**

- **query** – A reference to the query callable that returns the JSON content result to be wrapped.
- **bookmark** (*str*) – A string that enables you to specify which page of results you require. Only valid for queries using indexes of type *text*.
- **fields** (*list*) – A list of fields to be returned by the query.
- **page_size** (*int*) – Sets the page size for result iteration. Default is 100.
- **r** (*int*) – Read quorum needed for the result. Each document is read from at least 'r' number of replicas before it is returned in the results.
- **selector** (*str*) – Dictionary object describing criteria used to select documents.
- **sort** (*list*) – A list of fields to sort by. Optionally the list can contain elements that are single member dictionary structures that specify sort direction. For example `sort=['name', {'age': 'desc'}]` means to sort the query results by the

"name" field in ascending order and the "age" field in descending order.

- **use_index** (*str*) – Identifies a specific index for the query to run against, rather than using the Cloudant Query algorithm which finds what it believes to be the best index.

**__getitem__**(*arg*)

Provides QueryResult index access and index slicing support.

An `int` argument will be interpreted as a `skip` and then a get of the next document. For example `[100]` means skip the first 100 documents and then get the next document.

An `int` slice argument will be interpreted as a `skip:limit-skip` style pair. For example `[100:200]` means skip the first 100 documents then get up to and including the 200th document so that you get the range between the supplied slice values.

See *QueryResult* for more detailed index access and index slicing examples.

**Parameters** **arg** – A single value representing a key or a pair of values representing a slice. The argument value(s) must be `int`.

**Returns** Document data as a list in JSON format

**class** cloudant.result.**Result**(*method_ref*, ***options*)

Bases: `object`

Provides a key accessible, sliceable, and iterable interface to result collections. A Result object is constructed with a raw data callable reference such as the database API convenience method *all_docs()* or the View *__call__()* callable, used to retrieve data. A Result object can also use optional extra arguments for result customization and supports efficient, paged iteration over the result collection to avoid large result data from adversely affecting memory.

In Python, slicing returns by value, whereas iteration will yield elements of the sequence. This means that individual key access and slicing will perform better for smaller data collections, whereas iteration will be more efficient for larger data collections.

For example:

```
# Key access:

# Access by index value:
result = Result(callable)
result[9]                  # skip first 9 records and get 10th

# Access by key value:
result = Result(callable)
result['foo']              # get records matching 'foo'
result[ResultByKey(9)]     # get records matching 9

# Slice access:

# Access by index slices:
result = Result(callable)
result[100: 200]           # get records after the 100th and up to and including␣
↪the 200th
result[: 200]              # get records up to and including the 200th
result[100: ]              # get all records after the 100th
result[: ]                 # get all records

# Access by key slices:
result = Result(callable)
result['bar':'foo']        # get records between and including 'bar' and 'foo'
```

(continues on next page)

(continued from previous page)

```
result['foo':]            # get records after and including 'foo'
result[:'foo']            # get records up to and including 'foo'

result[['foo', 10]:
       ['foo', 11]]       # Complex key access and slicing works the same as␣
↪simple keys

result[ResultByKey(5):
       ResultByKey(10)]   # key slice access of integer keys

# Iteration:

# Iterate over the entire result collection
result = Result(callable)
for i in result:
    print i

# Iterate over the result collection between startkey and endkey
result = Result(callable, startkey='2013', endkey='2014')
for i in result:
    print i

# Iterate over the entire result collection in batches of 1000, including␣
↪documents.
result = Result(callable, include_docs=True, page_size=1000)
for i in result:
    print i
```

Note: Since Result object key access, slicing, and iteration use query parameters behind the scenes to handle their processing, some query parameters are not permitted as part of a Result customization, depending on whether key access, slicing, or iteration is being performed.

Such as:

| | |
|---|---|
| Access/Slicing by index value | No restrictions |
| Access/Slicing by key value | `key`, `keys`, `startkey`, `endkey` not permitted |
| Iteration | `limit`, `skip` not permitted |

**Parameters**

- **method_ref** (*str*) – A reference to the method or callable that returns the JSON content result to be wrapped as a Result.
- **descending** (*bool*) – Return documents in descending key order.
- **endkey** – Stop returning records at this specified key. Not valid when used with key access and key slicing.
- **endkey_docid** (*str*) – Stop returning records when the specified document id is reached.
- **group** (*bool*) – Using the reduce function, group the results to a group or single row.
- **group_level** – Only applicable if the view uses complex keys: keys that are JSON arrays. Groups reduce results for the specified number of array fields.
- **include_docs** (*bool*) – Include the full content of the documents.
- **inclusive_end** (*bool*) – Include rows with the specified endkey.

- **key** – Return only documents that match the specified key. Not valid when used with key access and key slicing.
- **keys** (*list*) – Return only documents that match the specified keys. Not valid when used with key access and key slicing.
- **limit** (*int*) – Limit the number of returned documents to the specified count. Not valid when used with key iteration.
- **page_size** (*int*) – Sets the page size for result iteration.
- **reduce** (*bool*) – True to use the reduce function, false otherwise.
- **skip** (*int*) – Skip this number of rows from the start. Not valid when used with key iteration.
- **stable** (*bool*) – Whether or not the view results should be returned from a "stable" set of shards.
- **stale** (*str*) – Allow the results from a stale view to be used. This makes the request return immediately, even if the view has not been completely built yet. If this parameter is not given, a response is returned only after the view has been built. Note that this parameter is deprecated and the appropriate combination of *stable* and *update* should be used instead.
- **startkey** – Return records starting with the specified key. Not valid when used with key access and key slicing.
- **startkey_docid** (*str*) – Return records starting with the specified document ID.
- **update** (*str*) – Determine whether the view in question should be updated prior to or after responding to the user. Valid values are: false: return results before updating the view; true: Return results after updating the view; lazy: Return the view results without waiting for an update, but update them immediately after the request.

**__getitem__**(*arg*)

Provides Result key access and slicing support.

An `int` argument will be interpreted as a `skip` and then a get of the next record. For example `[100]` means skip the first 100 records and then get the next record.

A `str`, `list` or *ResultByKey* argument will be interpreted as a `key` and then get all records that match the given key. For example `['foo']` will get all records that match the key 'foo'.

An `int` slice argument will be interpreted as a `skip:limit-skip` style pair. For example `[100:200]` means skip the first 100 records then get up to and including the 200th record so that you get the range between the supplied slice values.

A slice argument that contains `str`, `list`, or *ResultByKey* will be interpreted as a `startkey:endkey` style pair. For example `['bar': 'foo']` means get the range of records where the keys are between and including 'bar' and 'foo'.

See *Result* for more detailed key access and slicing examples.

**Parameters** **arg** – A single value representing a key or a pair of values representing a slice. The argument value(s) can be `int`, `str`, `list` (in the case of complex keys), or *ResultByKey*.

**Returns** Rows data as a list in JSON format

**__iter__**()

Provides iteration support, primarily for large data collections. The iterator uses the `skip` and `limit` options to consume data in chunks controlled by the `page_size` option. It retrieves a batch of data from the result collection and then yields each element.

See *Result* for Result iteration examples.

**Returns** Iterable data sequence

**all**()
Retrieve all results.

Specifying a `limit` parameter in the `Result` constructor will limit the number of documents returned. Be aware that the `page_size` parameter is not honoured.

**Returns** results data as list in JSON format.

**class** cloudant.result.**ResultByKey**(*value*)
Bases: `object`

Provides a wrapper for a value used to retrieve records from a result collection based on an actual document key value. This comes in handy when the document key value is an `int`.

For example:

```
result = Result(callable)
result[ResultByKey(9)]   # gets records where the key matches 9
# as opposed to:
result[9]                # gets the 10th record of the result collection

:param value: A value representing a Result key.
```

### 3.1.10 replicator

API module/class for handling database replications

**class** cloudant.replicator.**Replicator**(*client*)
Bases: `object`

Provides a database replication API. A Replicator object is instantiated with a reference to a client/session. It retrieves the `_replicator` database for the specified client and uses that database object to manage replications.

**Parameters** **client** – Client instance used by the database. Can either be a `CouchDB` or `Cloudant` client instance.

**create_replication**(*source_db=None*, *target_db=None*, *repl_id=None*, ***kwargs*)
Creates a new replication task.

**Parameters**

- **source_db** – Database object to replicate from. Can be either a `CouchDatabase` or `CloudantDatabase` instance.
- **target_db** – Database object to replicate to. Can be either a `CouchDatabase` or `CloudantDatabase` instance.
- **repl_id** (*str*) – Optional replication id. Generated internally if not explicitly set.
- **user_ctx** (*dict*) – Optional user to act as. Composed internally if not explicitly set.
- **create_target** (*bool*) – Specifies whether or not to create the target, if it does not already exist.
- **continuous** (*bool*) – If set to True then the replication will be continuous.

**Returns** Replication document as a Document instance

**follow_replication**(*repl_id*)
Blocks and streams status of a given replication.

For example:

```
for doc in replicator.follow_replication(repl_doc_id):
    # Process replication information as it comes in
```

**Parameters** **repl_id** (*str*) – Replication id used to identify the replication to inspect.

**Returns** Iterable stream of copies of the replication Document and replication state as a `str` for the specified replication id

**list_replications**()
Retrieves all replication documents from the replication database.

**Returns** List containing replication Document objects

**replication_state**(*repl_id*)
Retrieves the state for the given replication. Possible values are `triggered`, `completed`, `error`, and `None` (meaning not yet triggered).

**Parameters** **repl_id** (*str*) – Replication id used to identify the replication to inspect.

**Returns** Replication state as a `str`

**stop_replication**(*repl_id*)
Stops a replication based on the provided replication id by deleting the replication document from the replication database. The replication can only be stopped if it has not yet completed. If it has already completed then the replication document is still deleted from replication database.

**Parameters** **repl_id** (*str*) – Replication id used to identify the replication to stop.

### 3.1.11 feed

Module containing the Feed class which provides iterator support for consuming continuous and non-continuous feeds like `_changes` and `_db_updates`.

**class** cloudant.feed.**Feed**(*source*, *raw_data=False*, ***options*)
Bases: `object`

Provides an iterator for consuming client and database feeds such as `_db_updates` and `_changes`. A Feed object is constructed with a *client* or a *database* which it uses to issue HTTP requests to the appropriate feed endpoint. Instead of using this class directly, it is recommended to use the client APIs *db_updates()*, *db_updates()*, or the database API *changes()*. Reference those methods for a list of valid feed options.

**Parameters**

- **source** – Either a *client* object or a *database* object.
- **raw_data** (*bool*) – If set to True then the raw response data will be streamed otherwise if set to False then JSON formatted data will be streamed. Default is False.

**last_seq**
Returns the last sequence identifier for the feed. Only available after the feed has iterated through to completion.

**Returns** A string representing the last sequence number of a feed.

**next**()
Handles the iteration by pulling the next line out of the stream, attempting to convert the response to JSON if necessary.

**Returns** Data representing what was seen in the feed

**stop**()
Stops a feed iteration.

**class** cloudant.feed.**InfiniteFeed**(*source*, ***options*)
Bases: *cloudant.feed.Feed*

Provides an infinite iterator for consuming client and database feeds such as `_db_updates` and `_changes`. An InfiniteFeed object is constructed with a *Cloudant* object or a *database* object which it uses to issue HTTP requests to the appropriate feed endpoint. An infinite feed is NOT supported for use with a *CouchDB* object and unlike a *Feed* which can be a `normal`, `longpoll`, or `continuous` feed, an InfiniteFeed can only be `continuous` and the iterator will only stream formatted JSON objects. Instead of using this class directly, it is recommended to use the client API *infinite_db_updates()* or the database API `_infinite_changes()`. Reference those methods for a valid list of feed options.

Note: The infinite iterator is not exception resilient so if an unexpected exception occurs, the iterator will terminate. Any unexpected exceptions should be handled in code outside of this library. If you wish to restart the infinite iterator from where it left off that can be done by constructing a new InfiniteFeed object with the `since` option set to the sequence number of the last row of data prior to termination.

**Parameters** **source** – Either a *Cloudant* object or a *database* object.

**next**()
Handles the iteration by pulling the next line out of the stream and converting the response to JSON.

**Returns** Data representing what was seen in the feed

### 3.1.12 error

Module that contains common exception classes for the Cloudant Python client library.

**exception** cloudant.error.**CloudantArgumentError**(*code=100*, **args*)
Bases: *cloudant.error.CloudantException*

Provides a way to issue Cloudant Python client library specific exceptions that pertain to invalid argument errors.

Note: The intended use for this class is internal to the Cloudant Python client library.

**Parameters**

- **code** (*int*) – An optional code value used to identify the exception. Defaults to 100.
- **args** – A list of arguments used to format the exception message.

**exception** cloudant.error.**CloudantClientException**(*code=100*, **args*)
Bases: *cloudant.error.CloudantException*

Provides a way to issue Cloudant library client specific exceptions.

**Parameters**

- **code** (*int*) – A code value used to identify the client exception.
- **args** – A list of arguments used to format the exception message.

**exception** cloudant.error.**CloudantDatabaseException**(*code=100*, **args*)
Bases: *cloudant.error.CloudantException*

Provides a way to issue Cloudant library database specific exceptions.

**Parameters**

- **code** (*int*) – A code value used to identify the database exception.
- **args** – A list of arguments used to format the exception message.

**exception** cloudant.error.**CloudantDesignDocumentException**(*code=100*, **args*)
Bases: *cloudant.error.CloudantException*

Provides a way to issue Cloudant library design document exceptions.

**Parameters**

- **code** (*int*) – A code value used to identify the design doc exception.
- **args** – A list of arguments used to format the exception message.

**exception** cloudant.error.**CloudantDocumentException**(*code=100*, **args*)
Bases: *cloudant.error.CloudantException*

Provides a way to issue Cloudant library document specific exceptions.

**Parameters**

- **code** (*int*) – A code value used to identify the document exception.
- **args** – A list of arguments used to format the exception message.

**exception** cloudant.error.**CloudantException**(*msg*, *code=None*)
Bases: exceptions.Exception

Provides a way to issue Cloudant Python client library specific exceptions. A CloudantException object is instantiated with a message and optional code.

Note: The intended use for this class is internal to the Cloudant Python client library.

**Parameters**

- **msg** (*str*) – A message that describes the exception.
- **code** (*int*) – A code value used to identify the exception.

**exception** cloudant.error.**CloudantFeedException**(*code=100*, **args*)
Bases: *cloudant.error.CloudantException*

Provides a way to issue Cloudant library feed specific exceptions.

**Parameters**

- **code** (*int*) – A code value used to identify the feed exception.
- **args** – A list of arguments used to format the exception message.

**exception** cloudant.error.**CloudantIndexException**(*code=100*, **args*)
Bases: *cloudant.error.CloudantException*

Provides a way to issue Cloudant library index specific exceptions.

**Parameters**

- **code** (*int*) – A code value used to identify the index exception.
- **args** – A list of arguments used to format the exception message.

**exception** cloudant.error.**CloudantReplicatorException**(*code=100*, **args*)
Bases: *cloudant.error.CloudantException*

Provides a way to issue Cloudant library replicator specific exceptions.

**Parameters**

- **code** (*int*) – A code value used to identify the replicator exception.
- **args** – A list of arguments used to format the exception message.

**exception** cloudant.error.**CloudantViewException**(*code=100*, **args*)
Bases: *cloudant.error.CloudantException*

Provides a way to issue Cloudant library view specific exceptions.

**Parameters**

- **code** (*int*) – A code value used to identify the view exception.
- **args** – A list of arguments used to format the exception message.

**exception** cloudant.error.**ResultException**(*code=100*, **args*)
Bases: *cloudant.error.CloudantException*

Provides a way to issue Cloudant Python client library result specific exceptions.

**Parameters**

- **code** (*int*) – A code value used to identify the result exception. Defaults to 100.
- **args** – A list of arguments used to format the exception message.

### 3.1.13 adapters

Module that contains default transport adapters for use with requests.

**class** cloudant.adapters.**Replay429Adapter**(*retries=3*, *initialBackoff=0.25*)
Bases: requests.adapters.HTTPAdapter

A requests TransportAdapter that extends the default HTTPAdapter with configuration to replay requests that receive a 429 Too Many Requests response from the server. The duration of the sleep between requests will be doubled for each 429 response received.

Parameters can be passed in to control behavior:

**Parameters**

- **retries** (*int*) – the number of times the request can be replayed before failing.
- **initialBackoff** (*float*) – time in seconds for the first backoff.

- genindex

# Python Module Index

## C

# Index

## Symbols



## A



## B



## C

## D



## E



## F



## G

## S



## T



## U



## V