

---

# LightCouch User Guide 0.2.0

## Table of Contents

1. Overview .....	1
2. Setup .....	1
3. Setup (Android) .....	2
4. Configuration and Use .....	2
5. Documents API .....	4
5.1. Saving/Updating Documents .....	4
5.2. Finding Documents .....	5
5.3. Deleting Documents .....	6
5.4. Custom Serializers .....	6
6. Attachments API .....	6
6.1. Inline Attachments .....	6
6.2. Standalone Attachments .....	7
7. Logging and Error Handling .....	7
7.1. Logging .....	7
7.2. Error Handling .....	8
8. Views API .....	8
8.1. Special Views .....	9
8.2. Pagination .....	9
9. Bulk Documents .....	10
10. Design Documents .....	10
11. Database API .....	11
12. Change Notifications .....	12
13. Update Handlers .....	12
14. Replication .....	13
15. FullText Search .....	14
16. Extending the API .....	15

## 1. Overview

LightCouch provides a Java interface for communicating with CouchDB RESTful JSON APIs.

APIs in LightCouch is accessed by establishing a 'client' towards a CouchDB database server. A client is the main object which performs every request to the database, under it's context, such as document persistence or querying Views.

The APIs focus on Documents CRUD, Views, Attachments, Design Documents, Bulk operations, Changes notification, and database specific such as Compaction and Replication.

API usage and use cases examples is covered by integration unit tests, that runs against a running database. [Download](#).

Internally LightCouch depends on [Apache HttpClient](#) for handling the HTTP communications & [Gson](#) for JSON / Java mappings.

## 2. Setup

Maven

```
<dependency>
  <groupId>org.lightcouch</groupId>
  <artifactId>lightcouch</artifactId>
  <version>0.2.0</version>
</dependency>
```

Alternatively download the dependencies.

- [lightcouch-0.2.0.jar](#) [ *md5* ] - *sources*
- [HttpClient 4.5.3](#)
- [HttpCore 4.4.6](#)
- [Commons Codec 1.9](#)
- [Commons Logging 1.2](#)
- [Gson 2.8.2](#)

### 3. Setup (Android)

Include the following dependencies only, excluding HttpClient.

- [lightcouch-0.2.0.jar](#) [ *md5* ] - *sources*
- [Gson 2.8.2](#)

### 4. Configuration and Use

LightCouch client, i.e. [org.lightcouch.CouchDbClient](#) can be configured and constructed in several ways as preferred.

LightCouch could run on Android platform via [org.lightcouch.CouchDbClientAndroid](#)

Configuration parameters can be supplied to a client via `.properties` files, or through convenient constructors. Typical usage is creating a file named `couchdb.properties` in the default classpath of your application, containing the following parameters:

```
### Required
couchdb.name=db-test
couchdb.createdb.if-not-exist=true
# The protocol: http | https
couchdb.protocol=http
couchdb.host=127.0.0.1
# The port e.g: 5984 | 6984
couchdb.port=5984
# Blank username/password for no login
couchdb.username=
couchdb.password=

### Optional/Advanced
```

```
# Timeout to wait for a response in ms. Defaults to 0 (no timeout).
couchdb.http.socket.timeout=
# Timeout to establish a connection in ms. Defaults to 0 (no timeout).
couchdb.http.connection.timeout=
# Max connections.
couchdb.max.connections=100
# Connect through proxy
couchdb.proxy.host=
couchdb.proxy.port=
# path to append to DB URI
couchdb.path=
```

Next, create a client instance, using the default constructor:

```
CouchDbClient dbClient = new CouchDbClient();
```

`CouchDbClient` provides additional constructors for instantiating a client. One which accepts a `.properties` file name e.g, `mycouchdb.properties`, another that accepts individual parameters, the final uses an object with properties [CouchDbProperties](#).

```
// couchdb-2.properties is on the classpath
CouchDbClient dbClient1 = new CouchDbClient("couchdb-2.properties");

CouchDbClient dbClient2 = new CouchDbClient("db-name", true, "http", "127.0.0.1",

CouchDbProperties properties = new CouchDbProperties()
    .setDbName("db-name")
    .setCreateDbIfNotExist(true)
    .setProtocol("https")
    .setHost("example.com")
    .setPort(443)
    .setUsername("username")
    .setPassword("secret")
    .setMaxConnections(100)
    .setConnectionTimeout(0);

CouchDbClient dbClient3 = new CouchDbClient(properties);

// Client is ready to use
```

Typically a client instance is created once per an application, and reused.

Multiple client instances within an application; can be created to handle multiple databases, each instance runs in isolation.

After usage, it might be useful to shutdown a client's underlying connection manager, for proper release of resources:

```
dbClient.shutdown();
```

Example use in Spring application.

```
<bean id="dbClient" class="org.lightcouch.CouchDbClient"
      init-method="syncDesignDocsWithDb" destroy-method="shutdown"/>
```

```
@Autowired
private CouchDbClient dbClient;
```

## 5. Documents API

Documents API is accessed via a client instance which acts like JPA `EntityManager`.

The API provides CRUD kind of operations for documents.

Model classes that represent documents in CouchDB, can choose to extend [org.lightcouch.Document](#) for convenience. Plain objects need to define id and revision fields, i.e. `_id` & `_rev`.

Gson Mapping annotations eg. `@SerializedName("_id") String id;` could be used to indicate the JSON name of a field. Fields with null value are not serialized by default. An instance of Gson may be obtained by a client, for use in an application when desired.

### 5.1. Saving/Updating Documents

Saving / Updating Documents is handled by `save(Object)` and `update(Object)`.

An object can be:

1. Plain Java Object.
2. `java.util.Map`.
3. `com.google.gson.JsonObject`.

Saving a new object internally is handled by HTTP `PUT`, a case where a document must be assigned an id prior to sending to the database.

If the new object does not define an id, the API will generate a UUID similar to database generated. UUIDs can also be obtained from the database, an example below on Database API section.

Documents can also be saved via `batch(Object)`.

The Result of save or update operations returns a [org.lightcouch.Response](#) object that represents the database response.

```
CouchDbClient dbClient = new CouchDbClient();
Foo foo = new Foo();

Response response = dbClient.save(foo);
// dbClient.save(foo); // save, ignore response
// dbClient.batch(foo); // saves batch

dbClient.update(foo);
```

```
Map<String, Object> map = new HashMap<>();
map.put("_id", "doc-id");
map.put("title", "value");
dbClient.save(map);

JsonObject json = new JsonObject();
json.addProperty("_id", "doc-id");
json.add("array", new JsonArray());
dbClient.save(json);

String jsonstr = "{\"title\":\"val\"}";
JsonObject jsonobj = dbClient.getGson().fromJson(jsonstr, JsonObject.class);
dbClient.save(jsonobj);
```

## 5.2. Finding Documents

Finding documents API is available through `find()`s, that expect the document `id`, or `w/ rev`, and optional query parameters through [org.lightcouch.Params](#)

An additional finder `findAny()` gives the option to get any document from the database, given a URI, as string, encoding may be required.

Documents returned as:

1. Plain Java Objects.
2. `com.google.gson.JsonObject`.
3. `java.io.InputStream`.

`java.util.Map`, however does not qualify as a return type.

Note: Input streams need to be closed properly; to release the connection.

```
Foo foo = dbClient.find(Foo.class, "doc-id");
foo = dbClient.find(Foo.class, "doc-id", "doc-rev");

foo = dbClient.find(Foo.class, "doc-id", new Params().revsInfo().localSeq());

boolean found = dbClient.contains("doc-id");

InputStream in = dbClient.find("doc-id");
// ..
in.close(); // close stream

JsonObject json = dbClient.find(JsonObject.class, "doc-id");

String baseURI = dbClient.getBaseUri().toString();
String dbURI = dbClient.getDBUri().toString();

String uri = baseURI + "_stats";
```

```
JsonObject stats = dbClient.findAny(JsonObject.class, uri);
```

## 5.3. Deleting Documents

The API provides a `remove()` for deleting documents, expected parameters:

1. An object containing `_id` and `_rev`.
2. `id` and `revision` values.

```
Response response = dbClient.remove(object);  
  
dbClient.remove("doc-id", "doc-rev");
```

## 5.4. Custom Serializers

Configuring a custom serializer gives the option to register classes Gson may not be able to handle by default, eg. `JodaTime DateTime` class.

```
GsonBuilder gsonBuilder = new GsonBuilder();  
  
gsonBuilder.registerTypeAdapter(DateTime.class, new JsonSerializer<DateTime>() {  
    public JsonElement serialize(DateTime src, Type typeOfSrc,  
        JsonSerializationContext context) {  
        return new JsonPrimitive(src.toString());  
    }  
});  
  
gsonBuilder.registerTypeAdapter(DateTime.class, new JsonDeserializer<DateTime>() {  
    public DateTime deserialize(JsonElement json, Type typeOfT,  
        JsonDeserializationContext context) throws JsonParseException {  
        return new DateTime(json.getAsJsonPrimitive().getAsString());  
    }  
});  
  
CouchDbClient dbClient = new CouchDbClient();  
dbClient.setGsonBuilder(gsonBuilder);  
  
// .. client is ready to use
```

# 6. Attachments API

The API supports both inline and standalone Attachments.

## 6.1. Inline Attachments

[org.lightcouch.Attachment](#) represents an inline attachment enclosed in a document.

The base64 data of an attachment may be encoded utilizing the included dependency on Apache Codec `Base64.encodeBase64String(bytes)`.

Model classes that extend [org.lightcouch.Document](#) inherit the support for inline attachments.

To retrieve the base64 data of an attachment, include a parameter to the `find()`

```
byte[] bytes = "binary string".getBytes();
String data = Base64.encodeBase64String(bytes);

Attachment attachment = new Attachment(data, "text/plain");

Bar bar = new Bar();
bar.addAttachment("bar.txt", attachment);

dbClient.save(bar);

bar = dbClient.find(Bar.class, "doc-id", new Params().attachments());

String base64Data = bar.getAttachments().get("bar.txt").getData();
```

## 6.2. Standalone Attachments

Standalone attachments could be saved to an existing, or to a new document. The attachment data is provided as `java.io.InputStream`.

```
InputStream in = // .. init stream

// save under a new document, a generated UUID is assigned as id.
Response response = dbClient.saveAttachment(in, "photo.png", "image/png");

// save to an existing document
dbClient.saveAttachment(in, "file.pdf", "application/pdf", "doc-id", "doc-rev");

// save under a new document with the given id.
dbClient.saveAttachment(in, "photo.jpeg", "image/jpeg", "doc-id", null);

// get attachment
InputStream in = dbClient.find("doc-id/photo.jpeg");
// ..
in.close(); // close the stream
```

## 7. Logging and Error Handling

### 7.1. Logging

LightCouch logs information about API usage, such as a request URI, and the returned HTTP status code from the database server, such information is logged at the level `INFO`.

Log4j configuration:

```
# LightCouch
log4j.logger.org.lightcouch=INFO

# Apache HttpClient
log4j.logger.org.apache.http=ERROR
```

## 7.2. Error Handling

In the event of API call failure, an exception of a specific type is thrown to report the failure.

Defined Exceptions:

1. `NoDocumentException` when a requested document could not be found.
2. `DocumentConflictException` when a conflict is detected during a save or update.
3. `CouchDbException` is the super class for the former two exceptions, conveniently it is thrown in cases of unexpected exceptions, such as database errors 4xx or 5xx.

## 8. Views API

Views API is accessible under the context `view("design_name/my_view")`.

Views can be queried in a number of ways:

1. `List<T>` A list of specified type.
2. Scalar values, `int`, `boolean` etc.
3. View entries, key/value pairs as stored in the Database B-Tree.
4. `java.io.InputStream`.
5. Pagination.

```
// List<T>
List<Foo> list = dbClient.view("example/foo")
    .includeDocs(true)
    .startKey("start-key")
    .endKey("end-key")
    .limit(10)
    .query(Foo.class);

// primitive types
int count = dbClient.view("example/by_tag").key("couchdb").queryForInt();

// View entries keys, values and documents
View view = dbClient.view("example/by_date")
    .key(2011, 10, 15) // complex key as: values or array
    .reduce(false)
    .includeDocs(true);

ViewResult<int[], String, Foo> entries =
    view.queryView(int[].class, String.class, Foo.class);
```



## 8.1. Special Views

Views API extend to support special Views such as `_all_docs`.

```
// _all_docs
List<Document> allDocs = dbClient.view("_all_docs").query(Document.class);

// _all_docs paging
Page<JsonObject> allDocsPage = dbClient.view("_all_docs").queryPage(10, pageParam,
```

## 8.2. Pagination

Paging over a View results is offered by an API that features previous & next navigation model.

A View page is respresented by `Page<T>`

Usage example in web application.

```
// Servlet

public void doGet(HttpServletRequest request, HttpServletResponse r) {

    String param = request.getParameter("param");
    int resultsPerPage = 15;

    // null param gets the first page
    Page<Foo> page = dbClient.view("example/foo")
        .reduce(false)
        .queryPage(resultsPerPage, param, Foo.class);

    request.setAttribute("page", page);

    // .. forward to a JSP for display
}

<%-- JSP --%>

<%-- iterate over documents --%>
<c:forEach items="${page.resultList}" var="foo">
    <p>${foo.title}
</c:forEach>

<%-- show paging status --%>
<p>Showing:  ${page.resultFrom} - ${page.resultTo} of total ${page.totalResults}

<%-- handle navigation --%>
<p>
<c:choose>
```

```
<c:when test="{page.hasPrevious}">
  <a href="/getpage?param={page.previousParam}"> Previous </a>
</c:when>
<c:otherwise> Previous </c:otherwise>
</c:choose>

${page.pageNumber}

<c:choose>
  <c:when test="{page.hasNext}">
    <a href="/getpage?param={page.nextParam}"> Next </a>
  </c:when>
  <c:otherwise> Next </c:otherwise>
</c:choose>
```

## 9. Bulk Documents

Bulk documents API performs two operations: Modify & Fetch for bulk documents.

```
// insert/update docs
List<Object> newDocs = new ArrayList<>();
newDocs.add(new Foo());
newDocs.add(new JsonObject());

boolean newEdits = true;

List<Response> bulkResponse = dbClient.bulk(newDocs, newEdits);

// fetch multiple documents
List<String> keys = Arrays.asList(new String[]{"doc-id-1", "doc-id-2"});

List<Foo> docs = dbClient.view("_all_docs")
    .includeDocs(true)
    .keys(keys)
    .query(Foo.class);
```

## 10. Design Documents

The API for Design Documents is accessible under the context `design()`.

A Design Document is represented by [org.lightcouch.DesignDocument](#). Design Documents may be maintained on local `.js` files; saved in an application classpath.

An API provides a kind of synchronization to copy from local files to the database. [org.lightcouch.CouchDbDesign](#) lists all the supported operations.

An example design document can be found under the test package of the source code.

Design documents directory structure:

```
design-docs
  |- example
    |- filters
      |- filter1.js
      |- ..
    |- lists
      |- list1.js
      |- ..
    |- shows
      |- show1.js
      |- ..
    |- updates
      |- update1.js
      |- ..
    |- validate_doc_update
      |- validate_doc_update.js
    |- rewrites
      |- rewrites.js
    |- views
      |- by_date
        |- map.js
      |- by_tag
        |- map.js
        |- reduce.js
  |- example2
    ...
```

```
// find by dir name
DesignDocument designDoc = dbClient.design().getFromDesk("example");

dbClient.design().synchronizeWithDb(designDoc);

// find from Db
DesignDocument designDoc1 = dbClient.design().getFromDb("_design/example");

// synchronize all
dbClient.design().synchronizeAllWithDb();
// dbClient.syncDesignDocsWithDb(); // or use a client
```

## 11. Database API

Database specific APIs is accessible under the context `context()`.

```
CouchDbInfo dbInfo = dbClient.context().info();

String version = dbClient.context().serverVersion();

dbClient.context().compact();
```

```
dbClient.context().ensureFullCommit();

dbClient.context().createDB("new-db");

List<String> uuids = dbClient.context().uuids(100);
```

## 12. Change Notifications

Change notifications API is accessible under the context `changes()`. The API supports Changes feed of both types normal and continuous. A Row in the Changes feed is returned as `com.google.gson.JsonObject`

```
// get latest update seq
CouchDbInfo dbInfo = dbClient.context().info();
String since = dbInfo.getUpdateSeq();

// feed type normal
ChangesResult changes = dbClient.changes()
    .includeDocs(true)
    .limit(20)
    .since(since)
    .getChanges();

List<ChangesResult.Row> rows = changes.getResults();

for (ChangesResult.Row row : rows) {
    String docId = row.getId()
    JsonObject doc = row.getDoc();
}

// feed type continuous
Changes changes = dbClient.changes()
    .includeDocs(true)
    .since(since)
    .heartBeat(30000)
    .filter("example/filter")
    .continuousChanges();

// live access to continuous feeds
while (changes.hasNext()) {
    ChangesResult.Row feed = changes.next();

    String docId = feed.getId();
    JsonObject doc = feed.getDoc();

    // changes.stop(); // stop the running feed
}
```

## 13. Update Handlers

Update handler is invoked with `invokeUpdateHandler()`.

```
Params params = new Params()
    .addParam("field", "title")
    .addParam("value", "foo bar");

String output = dbClient.invokeUpdateHandler("example/updates", "doc-id", params);
```

Example Update Function:

```
design-docs
  |- example
    |- updates
      |- update1.js

function(doc, req) {
  var field = req.query.field;
  var value = req.query.value;
  var message = 'set '+field+' to '+value;
  doc[field] = value;
  return [doc, message];
}
```

## 14. Replication

Replication is handled by two sets of API, one that replicate by POSTing to `_replicate` URI, and the newer replicator database (i.e `_replicator`) introduced with CouchDB 1.1.0 .

```
// query params for filtered replication
// String queryParams = "{\"key1\":\"val\"}";
Map<String, Object> queryParams = new HashMap<>();
queryParams.put("key1", "val");

// normal replication
ReplicationResult result = dbClient.replication()
    .source("source-db")
    .target("http://user:secret@127.0.0.1:5984/target-db")
    .createTarget(true)
    .sinceSeq(100) // as of CouchDB 1.2
// .cancel() // cancel a replication
    .filter("example/filter1")
    .queryParams(queryParams);
    .trigger();

List<ReplicationHistory> histories = result.getHistories();

// _replicator database
Replicator replicator = dbClient.replicator()
    .source("source-db")
    .target("target-db")
    .continuous(true)
```

```
.createTarget(true)
.sinceSeq(100) // as of CouchDB 1.2
.replicatorDB("replicator_2") // optional, defaults to _replicator
.replicatorDocId("doc-id") // optional, defaults to UUID
.filter("example/filter1")
.queryParams(queryParams)
.userCtxName("user") // for delegated requests
.userCtxRoles("admin", "manager");

replicator.save(); // triggers a replication

ReplicatorDocument findDocument = dbClient.replicator()
    .replicatorDocId("doc-id")
    .find();

List<ReplicatorDocument> docs = dbClient.replicator().findAll();

dbClient.replicator()
    .replicatorDocId("doc-id")
    .replicatorDocRev("doc-rev")
    .remove(); // cancels an ongoing replication
```

## 15. FullText Search

FullText search is enabled by including the relevant data to design documents, then running the query using the usual API.

[couchdb-lucene](#) example:

```
design-docs
|- example
   |- fulltext
   |- fulltext.js

{
  "by_title": {
    "index": "function(doc) { var ret = new Document(); ret.add( doc.title ); r"
  }
}
```

```
String uri = dbClient.getBaseUri() + "_fti/local/mydb/_design/example/by_title?q=h
```

```
JsonObject result = dbClient.findAny(JsonObject.class, uri);
```

The second example uses [Cloudbant Search](#). Example design doc can be found [here](#).

```
design-docs
|- views101
```

```
|- indexes
  |- indexes.js
|- views
  |- ...
```

```
String uri = dbClient.getDBUri() + "_design/views101/_search/animals?q=kookaburra"

JsonObject result = dbClient.findAny(JsonObject.class, uri);
```

## 16. Extending the API

This API enables extending LightCouch internal API by allowing a user-defined raw HTTP request to execute against a database.

An example [get document current revision](#) is an API not provided by LightCouch.

```
import org.apache.http.client.methods.HttpHead;
import org.apache.http.HttpResponse;
import org.apache.http.client.utils.HttpClientUtils;

HttpHead head = new HttpHead(dbClient.getDBUri() + "doc-id");

HttpResponse response = dbClient.executeRequest(head);

String revision = response.getFirstHeader("ETAG").getValue();

HttpClientUtils.closeQuietly(response); // closes the response
```