PHPOnCouch Documentation

Release 2.0.3

Alexis Côté

October 01, 2017

Contents

CHAPTER 1

Overview

Introduction

PHPOnCouch tries to provide an easy way to work with your CouchDB documents with PHP .

What's new

Due to the lack of support on the last repository, I forked it and I will make sure it's kept active. Feel free to post any issue or feature request. I'm open for further developments but I don't have a lot of time.

With the new release of 2.0, the master branch will support only this version and the next one.

To access PHP-on-Couch for CouchDB 1.6.1, please visit this link.

Changelist

.. role:: hidden :class: hidden

2.0.4

Added

- Configuration
  - PHPOnCouch now supports configuration
  - You can now configure adapters via environment variables or .env files

Updated

- CouchClient
  - You can now pass credentials separately.
- Documentation
  - Moved the documentation to http://php-on-couch.readthedocs.io/

  - Translated documentation from Markdown to RST
  - Fixed few invalid documentation

.. role:: hidden :class: hidden

2.0.3

Added

- CouchClient
  - Added query parameters documentation for IDE
- CouchAdmin
  - setRolesToUser($user,$roles)
- Added missing tests for the library(code covered at 92%)
- Added detailed documentation for installation

Updated

- Fixed continuous stream (changes, continuous replication)
- Update code examples

.. role:: hidden :class: hidden

2.0.2

Added

- Couch
  - getAdapter()
  - setAdapter(CouchHttpAdapterInterface $adapter)
  - initAdapter($opts)
- Adapters
  - CouchHttpAdaterCurl
  - CouchHttpAdapterSocket
- doc/couch.md
- changelist.md

Fixed

- Removed echoes that were causing unexpected output
- Fixed some classes import
- Fixed Cookie parsing

.. role:: hidden :class: hidden

2.0.1

Added

- CouchClient
  - getIndexes()
  - createIndex(array $fields, $name = null, $ddoc = null, $type = 'json')
  - find($selector, array $fields = null, $sort = null, $index = null)
  - explain($selector, array $fields = null, $sort = null, $index = null)
- CouchClientTest
  - getIndexesTest()
  - createIndexTest()
  - findTest()
  - explainTest()
- changelist.md
- codestyle.md

Updated

- Refactored all the code to follow our code style
- Travis config to run CheckStyle
- Code example to correct syntax

Fixed

- Allow to use _users and _replicator databases directly

.. role:: hidden :class: hidden

2.0.0

Added

- CouchClient
  - getMemberShip()
  - getConfig($nodeName[,$section,$key])
  - setConfig($nodeName,$section,$key,$value)
  - deleteConfig($nodeName,$section,$key)
- Composer installation now available

Updated

- CouchAdmin($client,$options)
- Updated few tests cases

CHAPTER 2

Quickstart

Installation

To install the library and actually use it, you've got two choices:

- **Easy way** : Require the library and autoload it using Composer. This also make the updates way more easier and version.
- **Manual way** : Download the library from github and import the files from */src* to a *PHPOnCouch* folder in your project. You will need to update manually.

Version

Before we get into the installation, you need to know that multiple versions are available at the moment. Since it's forked project, you have always access to the origin branches. For the future, here are the description of the available versions :

- dev-master : The lastest tested sources
- 2.x.x : The PHP-on-Couch 2.0 Releases (Will be installed by default) This is the latest version that supports CouchDB 2.x.x
- 1.6.1.x-dev : The PHP-on-Couch 1.6.1 production branch. This branch contains the latest developments supporting CouchDB 1.6.1.

From this information, it is up to you to choose the right version. By default, the latest release will be installed.

Composer installation

Once you have composer installed, you are very close to have PHP-on-Couch installed. You simply need to do :

1. Add the root of your project, in a command shell, execute the following command : *composer require php-on-couch/php-on-couch*.

Note: By default, it will take the latest release*

2. Make sure your composer autoloader is called. If not, simply **require** the *autoload.php* file in *vendor* folder.
3. Start playing with PHPOnCouch!

Composer demo

The content pasted into the *index.php* file is :

```
<?php

//We need to use an autoloader to import PHPOnCouch classes
//I will use composer's autoloader for this demo
$autoloader = join(DIRECTORY_SEPARATOR,[__DIR__,'vendor','autoload.php']);
require $autoloader;

//We import the classes that we need
use PHPOnCouch\CouchClient;
use PHPOnCouch\Exceptions;

//We create a client to access the database
$client = new CouchClient('http://admin:adminPwd@localhost:5984','dbname');

//We create the database if required
if(!$client->databaseExists()){
    $client->createDatabase();
}

//We get the database info just for the demo
var_dump($client->getDatabaseInfos());

//Note:  Every request should be inside a try catch since CouchExceptions could be thrown.For example

try{
    $client->getDoc('the id');
    echo 'Document found';
}
catch(Exceptions\CouchNotFoundException $ex){
    if($ex->getCode() == 404)
        echo 'Document not found';
}
```

Manual installation

Since you have chose the manual installation, it's a bit more complicated but still simple! As you are probably reading this, you should be on Github. First of all, you need to select the branch that you want to install.

- Once you're on this branch, click the *Click or download* button and *Download ZIP*.
- Within the ZIP, extract the *src* folder into a folder named *PHPOnCouch* somewhere in your project.
- The only remaning step is to require the files. You can either use your own autloader or simply require the files manually.

Manual demo

index.php file content :

```
<?php

//We need to use an autoloader to import PHPOnCouch classes
//I will use PHPOnCouch autloader for the demo
```

```php
$autoloader = join(DIRECTORY_SEPARATOR,[__DIR__,'PHPOnCouch','autoload.php']);
require $autoloader;

//We import the classes that we need
use PHPOnCouch\CouchClient;
use PHPOnCouch\Exceptions;

//We create a client to access the database
$client = new CouchClient('http://admin:adminPwd@localhost:5984','dbname');

//We create the database if required
if(!$client->databaseExists()){
    $client->createDatabase();
}

//We get the database info just for the demo
var_dump($client->getDatabaseInfos());

//Note:  Every request should be inside a try catch since CouchExceptions could be thrown.For example

try{
    $client->getDoc('the id');
    echo 'Document found';
}
catch(Exceptions\CouchNotFoundException $ex){
    if($ex->getCode() == 404)
        echo 'Document not found';
}
```

And there you go! You can use the library from there following the *API*

Configuration

Since version 2.0.4, a configuration systeme has been implemented.

Configuration is made via environments variables. You can either:

- Define environments variables on your system
- Define configuration in a .env file directly at the root of the PHPOnCouch.

Configuration via file

In the *src* folder, you should see a file named: *.env.example*.

Copy the *.env.example* file to *.env*

Now, you can open the *.env* file and start configuration your library!

Configurations available

At the moment, you can configure curl options and the adapter used internally.

The HTTP adapter can either be curl or socket. By default, it will be curl if available.

If you are using a cURL adapter, you can specify CURL_OPT directly in the configuration file. Here's an example:

```
    CURLOPT_SSL_VERIFYPEER=1
```

Testing

To test the library, you needs two things:

- PHPUnit installed
- A CouchDB database running.

By default, the library is binded to "http://localhost:5984". You can change the host and the port by exporting ENV variables before running the tests.

Variables

Name	Type	Default	Description
DB_HOST	String	localhost	The host of the database (ip or dsn)
DB_PORT	Integer	5984	The port of the database. Note: for the moment, you can't change the port if you're using the docker image.

Install PHPUnit

The easy way to install PHPUnit is to use composer. In the project root, execute this :

```
composer install --dev
```

Run tests

Recommended way

PHP-on-Couch provides bash scripts that setup a CouchDB instance via docker and let you test the library. If you're on Windows, you have to install Git Bash which comes with Git when you install it.

The scripts for testing execute the following:

- Installs latest composer packages
- Starts the docker image
- Creates the databases required
- Seeds the database and setup users
- Runs the tests
- Validates the codestyle

Warning: Before running the scripts, make sure the port 5984 is free. Otherwise, the docker image won't be able to run and the tests will fail. Also, if you already have a local CouchDB, it's not recommended to use it for test. Tests will interact with the database and change it's current state.

For Windows users :

```
sh bin/_runLocalWin.sh
```

For Unix/OSX users :

```
sh bin/_runLocalUnix.sh
```

Contributing

Feel free to make any contributions. All contributions must follow the Coding Style Guide and must also comes with valid and complete tests.

Generating the documentation

To build the documentation, simply run the following script :

```
doc/make html
```

.. role:: hidden :class: hidden

Coding Style Guide

This guide is from PHP-FIG. The content is practically the same but we change few rules to be more flexible.

The code provided will be tested by the PHPCheckStyle tool as well as reviewed by the admin to be sure it's follows the project code style. You may contact the admins if you want to make suggestion to this guide.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1. Overview

- Code MUST follow a "coding style guide" PSR [PSR-1].
- Code MUST use 4 spaces for indenting, not tabs.
- There MUST NOT be a hard limit on line length; the soft limit MUST be 120 characters; lines SHOULD be 80 characters or less.
- There MUST be one blank line after the `namespace` declaration, and there MUST be one blank line after the block of `use` declarations.
- Opening braces for classes MUST go on the next line, and closing braces MUST go on the next line after the body.
- Opening braces for methods MUST go on the next line, and closing braces MUST go on the next line after the body.
- Visibility MUST be declared on all properties and methods; `abstract` and `final` MUST be declared before the visibility; `static` MUST be declared after the visibility.
- Control structure keywords MUST have one space after them; method and function calls MUST NOT.
- Opening braces for control structures MUST go on the same line, and closing braces MUST go on the next line after the body.

- Opening parentheses for control structures MUST NOT have a space after them, and closing parentheses for control structures MUST NOT have a space before.

1.1. Example

This example encompasses some of the rules below as a quick overview:

```
<?php
namespace Vendor\Package;

use FooInterface;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class Foo extends Bar implements FooInterface
{
    public function sampleMethod($a, $b = null)
    {
        if ($a === $b) {
            bar();
        } elseif ($a > $b) {
            $foo->bar($arg1);
        } else {
            BazClass::bar($arg2, $arg3);
        }
    }

    final public static function bar()
    {
        // method body
    }
}
```

1. General

2.1. Basic Coding Standard

Code MUST follow all rules outlined in PSR-1.

2.2. Files

All PHP files MUST use the Unix LF (linefeed) line ending.

All PHP files MUST end with a single blank line.

The closing `?>` tag MUST be omitted from files containing only PHP.

2.3. Lines

There MUST NOT be a hard limit on line length.

The soft limit on line length MUST be 120 characters; automated style checkers MUST warn but MUST NOT error at the soft limit.

Lines SHOULD NOT be longer than 80 characters; lines longer than that SHOULD be split into multiple subsequent lines of no more than 80 characters each.

There MUST NOT be trailing whitespace at the end of non-blank lines.

Blank lines MAY be added to improve readability and to indicate related blocks of code.

There MUST NOT be more than one statement per line.

2.4. Indenting

Code MUST use an indent of 4 spaces, and MUST NOT use tabs for indenting.

> N.b.: Using only spaces, and not mixing spaces with tabs, helps to avoid problems with diffs, patches, history, and annotations. The use of spaces also makes it easy to insert fine-grained sub-indentation for inter-line alignment.

2.5. Keywords and True/False/Null

PHP keywords MUST be in lower case.

The PHP constants `true`, `false`, and `null` MUST be in lower case.

1. Namespace and Use Declarations

When present, there MUST be one blank line after the `namespace` declaration.

When present, all `use` declarations MUST go after the `namespace` declaration.

There MUST be one `use` keyword per declaration.

There MUST be one blank line after the `use` block.

For example:

```
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

// ... additional PHP code ...
```

1. Classes, Properties, and Methods

The term “class” refers to all classes, interfaces, and traits.

4.1. Extends and Implements

The `extends` and `implements` keywords MUST be declared on the same line as the class name.

The opening brace for the class MUST go on its own line; the closing brace for the class MUST go on the next line after the body.

```
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements \ArrayAccess, \Countable
{
    // constants, properties, methods
}
```

Lists of `implements` MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one interface per line.

```
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements
    \ArrayAccess,
    \Countable,
    \Serializable
{
    // constants, properties, methods
}
```

4.2. Properties

Visibility MUST be declared on all properties.

The `var` keyword MUST NOT be used to declare a property.

There MUST NOT be more than one property declared per statement.

Property names SHOULD be prefixed with a single underscore to indicate private visibility.

A property declaration looks like the following.

```
<?php
namespace Vendor\Package;

class ClassName
{
    public $foo = null;
}
```

4.3. Methods

Visibility MUST be declared on all methods.

Method names SHOULD NOT be prefixed with a single underscore to indicate protected or private visibility.

Method names MUST be declared with a space after the method name. The opening brace MUST go on the same line, and the closing brace MUST go on the next line following the body. There MUST NOT be a space after the opening parenthesis, and there MUST NOT be a space before the closing parenthesis.

A method declaration looks like the following. Note the placement of parentheses, commas, spaces, and braces:

```
<?php
namespace Vendor\Package;

class ClassName {
    public function fooBarBaz($arg1, &$arg2, $arg3 = []) {
        // method body
    }
}
```

4.4. Method Arguments

In the argument list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

Method arguments with default values MUST go at the end of the argument list.

```
<?php
namespace Vendor\Package;

class ClassName {
    public function foo($arg1, &$arg2, $arg3 = []) {
        // method body
    }
}
```

Argument lists MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one argument per line.

When the argument list is split across multiple lines, the closing parenthesis and opening brace MUST be placed together on their own line with one space between them.

```
<?php
namespace Vendor\Package;

class ClassName {
    public function aVeryLongMethodName(
        ClassTypeHint $arg1,
        &$arg2,
        array $arg3 = []
    ) {
        // method body
    }
}
```

4.5. abstract, final, and static

When present, the abstract and final declarations MUST precede the visibility declaration.

When present, the static declaration MUST come after the visibility declaration.

```
<?php
namespace Vendor\Package;

abstract class ClassName
{
    protected static $foo;

    abstract protected function zim();

    final public static function bar()
    {
        // method body
    }
}
```

4.6. Method and Function Calls

When making a method or function call, there MUST NOT be a space between the method or function name and the opening parenthesis, there MUST NOT be a space after the opening parenthesis, and there MUST NOT be a space before the closing parenthesis. In the argument list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

```
<?php
bar();
$foo->bar($arg1);
Foo::bar($arg2, $arg3);
```

Argument lists MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one argument per line.

```
<?php
$foo->bar(
    $longArgument,
    $longerArgument,
    $muchLongerArgument
);
```

1. Control Structures

The general style rules for control structures are as follows:

- There MUST be one space after the control structure keyword
- There MUST NOT be a space after the opening parenthesis
- There MUST NOT be a space before the closing parenthesis
- There MUST be one space between the closing parenthesis and the opening brace
- The structure body MUST be indented once
- The closing brace MUST be on the next line after the body

The body of each structure MUST be enclosed by braces. This standardizes how the structures look, and reduces the likelihood of introducing errors as new lines get added to the body.

5.1. if, elseif, else

An `if` structure looks like the following. Note the placement of parentheses, spaces, and braces; and that `else` and `elseif` are on the same line as the closing brace from the earlier body.

```
<?php
if ($expr1) {
    // if body
} elseif ($expr2) {
    // elseif body
} else {
    // else body;
}
```

The keyword `elseif` SHOULD be used instead of `else if` so that all control keywords look like single words.

5.2. switch, case

A `switch` structure looks like the following. Note the placement of parentheses, spaces, and braces. The `case` statement MUST be indented once from `switch`, and the `break` keyword (or other terminating keyword) MUST be indented at the same level as the `case` body. There MUST be a comment such as `// no break` when fall-through is intentional in a non-empty `case` body.

```
<?php
switch ($expr) {
    case 0:
        echo `First case, with a break';
        break;
    case 1:
        echo `Second case, which falls through';
        // no break
    case 2:
    case 3:
    case 4:
        echo `Third case, return instead of break';
        return;
    default:
        echo `Default case';
        break;
}
```

5.3. while, do while

A `while` statement looks like the following. Note the placement of parentheses, spaces, and braces.

```
<?php
while ($expr) {
    // structure body
}
```

Similarly, a `do while` statement looks like the following. Note the placement of parentheses, spaces, and braces.

```
<?php
do {
    // structure body;
} while ($expr);
```

5.4. for

A `for` statement looks like the following. Note the placement of parentheses, spaces, and braces.

```
<?php
for ($i = 0; $i < 10; $i++) {
    // for body
}
```

5.5. foreach

A `foreach` statement looks like the following. Note the placement of parentheses, spaces, and braces.

```
<?php
foreach ($iterable as $key => $value) {
    // foreach body
}
```

5.6. try, catch

A `try catch` block looks like the following. Note the placement of parentheses, spaces, and braces.

```
<?php
try {
    // try body
} catch (FirstExceptionType $e) {
    // catch body
} catch (OtherExceptionType $e) {
    // catch body
}
```

1. Closures

Closures MUST be declared with a space after the `function` keyword, and a space before and after the `use` keyword.

The opening brace MUST go on the same line, and the closing brace MUST go on the next line following the body.

There MUST NOT be a space after the opening parenthesis of the argument list or variable list, and there MUST NOT be a space before the closing parenthesis of the argument list or variable list.

In the argument list and variable list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

Closure arguments with default values MUST go at the end of the argument list.

A closure declaration looks like the following. Note the placement of parentheses, commas, spaces, and braces:

```
<?php
$closureWithArgs = function ($arg1, $arg2) {
    // body
```

```
};

$closureWithArgsAndVars = function ($arg1, $arg2) use ($var1, $var2) {
    // body
};
```

Argument lists and variable lists MAY be split across multiple lines, where each subsequent line is indented once. When doing so, the first item in the list MUST be on the next line, and there MUST be only one argument or variable per line.

When the ending list (whether of arguments or variables) is split across multiple lines, the closing parenthesis and opening brace MUST be placed together on their own line with one space between them.

The following are examples of closures with and without argument lists and variable lists split across multiple lines.

```
<?php
$longArgs_noVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) {
    // body
};

$noArgs_longVars = function () use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
    // body
};

$longArgs_longVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
    // body
};

$longArgs_shortVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use ($var1) {
    // body
};

$shortArgs_longVars = function ($arg) use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
```

```
) {
    // body
};
```

Note that the formatting rules also apply when the closure is used directly in a function or method call as an argument.

```
<?php
$foo->bar(
    $arg1,
    function ($arg2) use ($var1) {
        // body
    },
    $arg3
);
```

1. Conclusion

There are many elements of style and practice intentionally omitted by this guide. These include but are not limited to:

- Declaration of global variables and global constants
- Declaration of functions
- Operators and assignment
- Inter-line alignment
- Comments and documentation blocks
- Class name prefixes and suffixes
- Best practices

Future recommendations MAY revise and extend this guide to address those or other elements of style and practice.

CHAPTER 3

API

Couch class

Summary

The Couch.php class is the one of the low level class that is used to handle the communication between the high level classes and CouchDB. Before version **2.0.2**, the default Http adapter was curl and all the possible adapters where declared into the Couch.php class. With **2.0.2**, the library code has been refactored so that the Http adapters are declared into separate classes. The Couch class nowaday use a HttpAdapterInterface to communicate with CouchDB.

Note: The following methods are public methods of the Couch class. Therefore, you will mostly use the high level classes which usually inherit the Couch class. For example, all the following methods will be directly available from the CouchClient class.

API Reference

class PHPOnCouch**Couch**
This is the low level class that handles communications with CouchDB.

dsn

PHPOnCouch\Couch::**dsn**()

Returns The dsn of the current Couch instance

options

PHPOnCouch\Couch::**options**()

Returns The options passed to the Couch instance.

getSessionCookie

PHPOnCouch\Couch::**getSessionCookie**()

Returns The current session cookie. Returns null if not set.

setSessionCookie

PHPOnCouch\Couch::**setSessionCookie**(*$cookie*)
Set the current session cookie.

Params string $cookie The cookie to set

query

PHPOnCouch\Couch::**query**(*$method*, *$url*, *$parameters = array()*, *$data = null*, *$contentType = null*)
Send a query to the CouchDB server.

Params string $method The HTTP method to use (GET,PUT,POST,...)

Params string $url The URL to fetch

Params array $parameters The query parameters to pass to the query

Params mixed $data The request body(null by default)

Params string $contentType The content type of the data.

Returns The server response or false if an error occured.

continuousQuery

PHPOnCouch\Couch::**continuousQuery**(*$callable*, *$method*, *$url*, *$parameters = array()*, *$data = null*)
Send a query to CouchDB. For each line returned by the server, the $callable will be called. If the callable returns false, the **continuousQuery** will stop.

Params Function $callable The function called for every document returned.

Params string $method The HTTP method to use (GET,PUT,POST,...)

Params string $url The URL to fetch

Params array $parameters The query parameters to pass to the query

Params mixed $data The request body(null by default)

Params string $contentType The content type of the data.

Returns The server response or false if an error occured.

storeFile

PHPOnCouch\Couch::**storeFile**(*$url*, *$file*, *$contentType*)
Make a request with the $file content passed into the request body. The $file must be on the disk.

Params function $callable The function called for every document returned

Params string $method The HTTP method to use (GET,PUT,POST,...)

Params string $url The URL to fetch

Params array $parameters The query parameters to pass to the query

Params mixed $data The request body(null by default)

Params string $contentType The content type of the data.

Returns The server response or false if an error occured.

storeAsFile

PHPOnCouch\Couch::**storeAsFile**(*$url*, *$data*, *$contentType*)

Make a request with the $data passed into the request body.

Params function $callable The function called for every document returned

Params string $method The HTTP method to use (GET,PUT,POST,...)

Params string $url The URL to fetch

Params array $parameters The query parameters to pass to the query

Params mixed $data The request body(null by default)

Params string $contentType The content type of the data.

Returns The server response or false if an error occured.

initAdapter

PHPOnCouch\Couch::**initAdapter**(*$options*)

This function is called to initialized the adapter. By default, it will load the cURL adapter. The options passed are the same options passed to the Couch class. It's must be an array of options. **You don't have to call this method.** It will be automatically call when using the Couch class.

Params array $options The options passed to the Couch instance

Example :

```
$couch = new Couch("http://localhost:5984");
$couch->initAdapter([]) //Set the curl by default
```

getAdapter

PHPOnCouch\Couch::**getAdapter**()

This function return the current adapter. If it's not set, the *Couch::initAdapter* will be called.

Returns The Adapter currently used.

Example :

```
$couch = new PHPOnCouch\Couch("http://localhost:5984");
$adapter = $couch->getAdapter();
$doc =  $adapte->query('GET','db/_all_docs');
```

setAdapter

PHPOnCouch\Couch::**setAdapter**(*CouchHttpAdapterInterface $adapter*)

This function set the current adapter of the Couch class. You must specify a class that implements the CouchHttpAdapterInterface.

Params CouchHttpAdapterInterface $adapter The adapter to set.

You can implemented the following adapters :

- CouchHttpAdapterSocket

•CouchHttpAdapterCurl (default)

Note: Even if the CouchHttpAdapter used is Curl, the Socket adapter is still used for the continuous_query function since it is not implemented with cURL.

Example:

```
use PHPOnCouch\Adapter\CouchHttpAdapterCurl;

$couch = new PHPOnCouch\Couch("http://localhost:5984");
$adapter = new CouchHttpAdapterSocket([]);
$couch->setAdapter($adapter);
```

CouchClient class

Database

This section give details on actions on the CouchDB server through PHP on Couch.

Getting started

To use PHP on Couch client, you have to create a couchClient instance, setting the URL to your couchDB server, and the database name.

Example : connect to the couchDB server at http://my.server.com on port 5984 and on database mydb :

```
$client = new CouchClient("http://my.server.com:5984/","mydb");
```

If you want to authenticate to the server using a username & password, just set it in the URL.

Example : connect to the couchDB server at http://my.server.com on port 5984 using the username "couchAdmin", the password "secret" and on database mydb :

```
$client = new CouchClient("http://couchAdmin:secret@my.server.com:5984/","mydb");
```

You can also tell couchClient to use cookie based authentification, by passing an additional flag "cookie_auth" set to true in the options array, as the third parameter of the couchClient constructor.

Example : as the previous one, but using cookie based authentification

```
$client = new CouchClient("http://couchAdmin:secret@my.server.com:5984/","mydb", array("cookie_auth"=
```

You can also manually set the session cookie.

Example : manually setting the session cookie :

```
$client = new CouchClient("http://my.server.com:5984/","mydb");
$client->setSessionCookie("AuthSession=Y291Y2g6NENGNDgzNzY6Gk0NjM-UKxhpX_IyiH-C-9yXY44");
```

General functions

class PHPOnCouch**CouchClient**

__construct

PHPOnCouch\CouchClient::**__construct**(*$dsn*, *$dbname*, *$options* =[ ])

Params string The complete url to the host. You can enter the credentials directly in it if they don't required to be encoded.

Params string The database name to use

Params array An array of options that can be pass. You can pass the following parameters : username, password, cookie_auth.

You can pass credentials to be encoded correctly. Example:

```
$client = new CouchClient('http://localhost:5984/','mydb',['username'=>'myuser','password'=>
```

You can also specify to use the cookie authentification by passing the 'cookie_auth' key.

Example:

```
$client = new CouchClient('http://localhost:5984/','mydb',['cookie_auth'=>true]);
$cookie = $client->getSessionCookie();
```

dsn

PHPOnCouch\CouchClient::**dsn**()

Returns string The DSN of the server. Database name is not included.

Example :

```
$client = new CouchClient("http://couch.server.com:5984/","hello");
echo $client->dsn(); // will echo : http://couch.server.com:5984
```

getSessionCookie

PHPOnCouch\CouchClient::**getSessionCookie**()

Returns string Returns the current session cookie if set.

Example :

```
$cookie = $client->getSessionCookie();
```

setSessionCookie

PHPOnCouch\CouchClient::**setSessionCookie**(*$cookie*)

This method set the cookie and is chainable.

Params string $cookie The cookie to set.

Returns CouchClient Return the current instance.

Example :

```
$cookie = $client->setSessionCookie("AuthSession=Y291Y2g6NENGNDgzNz")->getSessionCookie();
```

isValidaDatabaseName

PHPOnCouch\CouchClient::**isValidDatabaseName**(*$name*)

Database names on CouchDB have restrictions. Here are the allowed characters:

- lowercase characters (a-z)
- digits (0-9)
- any of the following characters _, $, (,), +, -, and / are allowed

The name has to start with a lowercase letter (a-z) or an underscore (_).

To test if a given database name is valid, use the static **isValidDatabaseName()** CouchClient method.

Params string $name The name to validate.

Returns boolean True if valid. Otherwise false.

Example :

```
$my_database = "user311(public)";
if ( CouchClient::isValidDatabaseName($my_database) ) {
    $client = new CouchClient("http://couch.server.com:5984/",$my_database);
} else {
    die("Invalid database name");
}
```

listDatabases

PHPOnCouch\CouchClient::**listDatabases**()

The method **listDatabases()** lists the available databases on the CouchDB server.

Returns array An array of database names.

Example :

```
$dbs = $client->listDatabases();
print_r($dbs); // array ('first_database','another_database')
```

createDatabase

PHPOnCouch\CouchClient::**createDatabase**()

Create the database according to the name you set when creating couch_client object $client.

Note: If the database already exist, this method will throw an exception.

Example :

```
$client->createDatabase();
```

deleteDatabase

PHPOnCouch\CouchClient::**deleteDatabase**()

Permanently remove from the server the database according to the name you set when creating couch_client object $client.

Note: If the database does not exist, the method will throw an exception.

Example :

```
$client->deleteDatabase();
```

databaseExists

PHPOnCouch\CouchClient::**databaseExists**()

Test if the database already exist on the server.

Returns boolean True if it exists. Otherwise false.

Example :

```
if ( !$client->databaseExists() ) {
    $client->createDatabase();
}
```

getDatabaseInfos

PHPOnCouch\CouchClient::**getDatabaseInfos**()

Sends back informations about the database. Informations contains the number of documents in the database, the space of the database on disk, the update sequence number, ...

Returns array Returns an arrayf with the database informations.

Example :

```
print_r($client->getDatabaseInfos());
/*
array("db_name" => "testdb",
      "doc_count" => 2,
      "doc_del_count" => 0,
      "update_seq" => 6,
      "purge_seq" => 0,
      "compact_running" => false,
      "disk_size" => 277707,
      "instance_start_time" => "1246277543362647"
 )
 */
```

getDatabaseUri

PHPOnCouch\CouchClient::**getDatabaseUri**()

The method **getDatabaseUri()** sends back a string giving the HTTP connection URL to the database server.

Example :

```
echo $client->getDatabaseUri();
/*
http://db.example.com:5984/testdb
*/
```

getUuids

PHPOnCouch\CouchClient::**getUuids**(*$count = 1*)

Sends back an array of universally unique identifiers (that is, big strings that can be used as document ids)

Params int $count The number of id to returns.

Returns array An array of identifiers

Example :

```
print_r($client->getUuids(5));
/*
    array ( 0 => "65a8f6d272b3e5e62ee9de8eacc083a5",
            1 => "e43b04e44233d72b353c1cd8915b886d",
            2 => "7498fb296f19ebc2554a4812f3d9ae12",
            3 => "f3f855a15eb90e9fcdbda5e017b9f2cd",
            4 => "9d9a8214762d06cdf0158d7f6697cac9" )
*/
```

useDatabase

PHPOnCouch\CouchClient::**useDatabase**(*$dbName*)

The method **useDatabase($dbname)** changes the working database on the CouchDB server.

Params string $dbName The name of the database to use.

Example :

```
$client = new CouchClient("http://localhost:5984", "db1");
$all_docs_db1 = $client->getAllDocs(); //retrieve all docs of database db1
$client->useDatabase("db2");           //switch to "db2" database
$all_docs_db2 = $client->getAllDocs(); //retrieve all docs of database db2
```

getMembership

PHPOnCouch\CouchClient::**getMembership**()

With the new Cluster infrastructure in CouchDB 2.0, you now have to configure each nodes. To do so, you need to get the information about them. The *_membership* endpoint allow you to get all the nodes that the current nodes knows and all the nodes that are in the same cluster. The method **getMembership()** returns an object like this :

```
{
  "all_nodes": [],
  "cluster_nodes": []
}
```

getConfig

PHPOnCouch\CouchClient::**getConfig**(*$nodeName*[, *$section*[, *$key*]])

> **Warning:** The configurations methods are implemented for PHP-on-Couch 2.0 only. Note that the configuration is per-node only.

To configure, you need to use **getConfig($nodeName [, $section [, $key]])**. If you don't know the nodeName, you can use the **getMembership()** method.

Params string $nodeName The name of the node to use.

Params string $section The section value to return.

Params string $key The section key-value to return.

Examples :

*getConfig("couchdb@localhost")**

Returns a JSON object with the whole configuration

```
{
    "attachments":{

    },
    "couchdb":{

    }
}
```

getConfig("couchdb@localhost","httpd")

Note: It will return a CouchNotFoundException is the section is not present.

Returns a JSON object that represent the desired section

```
{
    "allow_jsonp": "false",
    "authentication_handlers": "{couch_httpd_oauth, oauth_authentication_handler}, {couch_ht
    "bind_address": "127.0.0.1",
    "default_handler": "{couch_httpd_db, handle_request}",
    "enable_cors": "false",
    "log_max_chunk_size": "1000000",
    "port": "5984",
    "secure_rewrites": "true",
    "vhost_global_handlers": "_utils, _uuids, _session, _oauth, _users"
}
```

getConfig("couchdb@localhost","log","level")

Returns either text-plain of JSON value of the section/key.

Note: It will return a CouchNotFoundException is the section or key are not present*.

```
"debug"
```

setConfig

PHPOnCouch\CouchClient::**setConfig**(*$nodeName*, *$section*, *$key*, *$value*)

Warning: The configurations methods are implemented for PHP-on-Couch 2.0 only. Note that the configuration is per-node only*

The method **setConfig($nodeName, $section, $key, $value)** let you configure your installation. It can throws CouchNotAuthorizedException or CouchNotFoundException depending on the parameters supplied.

Example :

```
$val = $client->setConfig("couchdb@localhost","log","level","info");
echo $val;
/*
"debug"
*/
```

deleteConfig

PHPOnCouch\CouchClient::**deleteConfig**(*$nodeName*, *$section*, *$key*)

> **Warning:** The configurations methods are implemented for PHP-on-Couch 2.0 only. Note that the configuration is per-node only

The method **deleteConfig($nodeName, $section, $key)** let you delete a configuration key from your node. It will returns the JSON value of the parameter before its deletion. Not that the method can throw a CouchNotFoundException or a CouchUnauthorizedException regarding of the section/key and permissions.

Example:

```
$oldValue = $client->deleteConfig("couchdb@localhost","log","level");
echo $oldValue;
/*
"info"
*/
```

Changes

CouchDB implements database changes feedback and polling. You'll find more infos here . For any event in the database, CouchDB increments a sequence counter.

getChanges

PHPOnCouch\CouchClient::**getChanges**()

The method **getChanges()** sends back a CouchDB changes object.

Example :

```
print_r($client->getChanges());
/*
    stdClass Object
    (
        [results] => Array
            (
                [0] => stdClass Object
                    (
                        [seq] => 'example-last-update-sequence'
                        [id] => 482fa0bed0473fd651239597d1080f03
                        [changes] => Array
```

```
                                    (
                                        [0] => stdClass Object
                                            (
                                                [rev] => 3-58cae2758cea3e82105e1090d81a9e02
                                            )

                                    )

                                [deleted] => 1
                            )

                        [1] => stdClass Object
                            (
                                [seq] => 'example-last-update-sequence'
                                [id] => 2f3f913f34d60e473fad4334c13a24ed
                                [changes] => Array
                                    (
                                        [0] => stdClass Object
                                            (
                                                [rev] => 1-4c6114c65e295552ab1019e2b046b10e
                                            )

                                    )

                            )

                    )

                [last_seq] => 4
            )
        */
```

Chainable methods to use before getChanges()

The following methods allow a fine grained control on the **changes** request to issue.

since

PHPOnCouch\CouchClient::**since**(*string $value*)
Retrieve changes that happened after sequence number $value

Params string $value The minimal sequence number

heartbeat

PHPOnCouch\CouchClient::**heartbeat**(*integer $value*)

Params integer $value Number of milliseconds between each heartbeat line (an ampty line) one logpoll and continuous feeds

feed

PHPOnCouch\CouchClient::**feed**(*string $value*, *$callback*)
Feed type to use. In case of "continuous" feed type, $callback should be set and should be a PHP

callable object (so *is_callable($callback)* should be true)

The callable function or method will receive two arguments : the JSON object decoded as a PHP object, and a couchClient instance, allowing developers to issue CouchDB queries from inside the callback.

Params string $value The feed value.

Params callable $callback The callback function to execute for each document received.

filter

PHPOnCouch\CouchClient::**filter**(*string $value*, *array $additional_query_options*)
Apply the changes filter $value. Add additional headers if any

Params string $value The filter to use.

Params array $additional_query_options The additional query options to pass to the filter.

style

PHPOnCouch\CouchClient::**style**(*string $value*)
Changes display style, use "all_docs" to switch to verbose

Params string $value The style to value to apply

Example :

```
// fetching changes since sequence 'example-last-update-sequence' using filter "messages/inc
$changes = $client->since('example-last-update-sequence')->filter("messages/incoming")->getC
```

Example - Continuous changes with a callback function

```
function index_doc($change,$couch) {
    if( $change->deleted == true ) {
        // won't index a deleted file
        return ;
    }
    echo "indexing ".$change->id."\n";
    $doc = $couch->getDoc($change->id);
    unset($doc->_rev);
    $id = $doc->_id;
    unset($doc->_id);
    my_super_fulltext_search_appliance::index($id, $doc);
}

$client->feed('continuous','index_doc')->getChanges();
// will return when index_doc returns false or on socket error
```

ensureFullCommit

PHPOnCouch\CouchClient::**ensureFullCommit**()
The method **ensureFullCommit()** tells couchDB to commit any recent changes to the database file on disk.

Example :

```
$response = $client->ensureFullCommit();
print_r($response);
/* should print something like :
 stdClass Object
    (
        [ok] => 1,
        [instance_start_time] => "1288186189373361"
    )
*/
```

Maintenance

Three main maintenance tasks can be performed on a CouchDB database : compaction, view compaction, and view cleanup.

compactDatabase

PHPOnCouch\CouchClient::**compactDatabase**()

CouchDB database file is an append only : during any modification on database documents (add, remove, or update), the modification is recorded at the end of the database file. The compact operation removes old versions of database documents, thus reducing database file size and improving performances. To initiate a compact operation, use the **compactDatabase()** method.

Example :

```
// asking the server to start a database compact operation
$response = $client->compactDatabase(); // should return stdClass ( "ok" => true )
```

compactAllViews

PHPOnCouch\CouchClient::**compactAllViews**()

Just as documents files, view files are also append-only files. To compact all view files of all design documents, use the **compactAllViews()** method.

Example :

```
// asking the server to start a view compact operation on all design documents
$response = $client->compactAllViews(); // return nothing
```

compactViews

PHPOnCouch\CouchClient::**compactViews**(*$id*)

To compact only views from a specific design document, use the **compactViews($id)** method.

Params string $id The id of the design document to compact.

Example :

```
// asking the server to start a database compact operation on the design document _design/ex
$response = $client->compactViews( "example" ); // should return stdClass ( "ok" => true )
```

cleanupDatabaseViews

PHPOnCouch\CouchClient::**cleanupDatabaseViews**()

This operation will delete all unused view files. Use the **cleanupDatabaseViews()** method to initiate a cleanup operation on old view files

Example :

```
// asking the server to start a database view files cleanup operation
$response = $client->cleanupDatabaseViews(); // should return stdClass ( "ok" => true )
```

Document

class PHPOnCouch**CouchClient**

getAllDocs

PHPOnCouch\CouchClient::**getAllDocs**()

Retrieve all documents from the database. In fact it only retrieve document IDs, unless you specify the server to include the documents using the View query parameters syntax.

Returns An object with the total_rows number, the rows returned and other informations.

Example :

```
$all_docs = $client->getAllDocs();
echo "Database got ".$all_docs->total_rows." documents.<BR>\n";
foreach ( $all_docs->rows as $row ) {
    echo "Document ".$row->id."<BR>\n";
}
```

getDoc

PHPOnCouch\CouchClient::**getDoc**(*$id*)

Gives back the document that got ID $id, if it exists. Note that if the document does not exist, the method will throw an error.

Params string $id The id of the document to fetch

Returns The document if found. Otherwise, a CouchNotFoundException is throw.

The document is sent back as an HTTP object of class stdClass.

Example :

```
try {
    $doc = $client->getDoc("some_doc_id");
} catch ( Exception $e ) {
    if ( $e->getCode() == 404 ) {
       echo "Document some_doc_id does not exist !";
        }
    exit(1);
}
echo $doc->_id.' revision '.$doc->_rev;
```

Chainable methods to use with getDoc()

rev

PHPOnCouch\CouchClient::**rev**(*$value*)

The chainable **rev($value)** method specify the document revision to fetch.

Params mixed $value The specific revision to fetch of a document.

Returns The CouchClient instance.

Example :

```
try {
    $doc = $client->rev("1-849aff6ad4a38b1225c80a2119dc31cb")->getDoc("some_doc_id");
} catch ( Exception $e ) {
    if ( $e->getCode() == 404 ) {
       echo "Document some_doc_id or revision 1-849aff6ad4a38b1225c80a2119dc31cb does not ex
    }
    exit(1);
}
echo $doc->_rev ; // should echo 1-849aff6ad4a38b1225c80a2119dc31cb
```

asCouchDocuments

PHPOnCouch\CouchClient::**asCouchDocuments**()

When the getDoc function will be called, it will return a *CouchDocument*. You can however get back the document as a CouchDocument object by calling the **asCouchDocuments()** method before the **getDoc($id)** method.

Returns The CouchClient instance.

Example :

```
try {
    $doc = $client->asCouchDocuments()->getDoc("some_doc_id");
} catch ( Exception $e ) {
    if ( $e->getCode() == 404 ) {
       echo "Document some_doc_id does not exist !";
    }
    exit(1);
}
echo get_class($doc); // should echo "CouchDocument"
```

conflicts

PHPOnCouch\CouchClient::**conflicts**()

The chainable method **conflicts()** asks CouchDB to add to the document a property *_conflicts* containing conflicting revisions on an object.

Returns The CouchClient instance.

Example :

```
try {
    $doc = $client->conflicts()->getDoc("some_doc_id");
} catch ( Exception $e ) {
    if ( $e->getCode() == 404 ) {
```

```
            echo "Document some_doc_id does not exist !";
        }
        exit(1);
    }
    if ( $doc->_conflicts ) {
        print_r($doc->_conflicts);
    }
```

revs

PHPOnCouch\CouchClient::**revs**()

The chainable method **revs()** asks CouchDB to add to the document a property *_revisions* containing the list of revisions for an object.

Returns The CouchClient instance.

Example :

```
    try {
        $doc = $client->revs()->getDoc("some_doc_id");
    } catch ( Exception $e ) {
        if ( $e->getCode() == 404 ) {
            echo "Document some_doc_id does not exist !";
        }
        exit(1);
    }
    print_r($doc->_revisions);
```

revs_info

PHPOnCouch\CouchClient::**revs_info**()

The chainable method **revs_info()** asks CouchDB to add to the document a property *_revs_info* containing the avaibility of revisions for an object.

Returns The CouchClient instance.

Example :

```
    try {
        $doc = $client->revs_info()->getDoc("some_doc_id");
    } catch ( Exception $e ) {
        if ( $e->getCode() == 404 ) {
            echo "Document some_doc_id does not exist !";
        }
        exit(1);
    }
    print_r($doc->_revs_info);
```

open_revs

PHPOnCouch\CouchClient::**open_revs**(*$value*)

Using the **open_revs($value)** method, CouchDB returns an array of objects.

Params array|string $value Should be an array of revision ids or the special keyword all (to fetch all revisions of a document)

Returns The CouchClient instance.

Example :

```
try {
    $doc = $client->open_revs( array("1-fbd8a6da4d669ae4b909fcdb42bb2bfd", "2-5bc3c6319edf62
} catch ( Exception $e ) {
    if ( $e->getCode() == 404 ) {
       echo "Document some_doc_id does not exist !";
    }
    exit(1);
}
print_r($doc->_revs_info);
```

Which should return something similar to :

```
array (
    stdClass(
        "missing" => "1-fbd8a6da4d669ae4b909fcdb42bb2bfd"
    ),
    stdClass(
        "ok" => stdClass(
            "_id"  => "some_doc_id",
            "_rev" => "2-5bc3c6319edf62d4c624277fdd0ae191",
            "hello"=> "foo"
        )
    )
)
```

storeDoc

PHPOnCouch\CouchClient::**storeDoc**(*$doc*)

Store a document on the CouchDB server.

Params stdClass $doc $doc should be an object. If the property $doc->_rev is set, the method understand that it's an update, and as so requires the property $doc->_id to be set. If the property $doc->_rev is not set, the method checks for the existance of property $doc->_id and initiate the appropriate request.

Returns The response of this method is the CouchDB server response. In other words if the request ends successfully the returned object should be :

```
stdClass ( "ok" => true, "id" => "some_doc_id" , "rev" => "3-23423423476" )
```

Example : creating a document without specifying id

```
$new_doc = new stdClass();
$new_doc->title = "Some content";
try {
    $response = $client->storeDoc($new_doc);
} catch (Exception $e) {
    echo "ERROR: ".$e->getMessage()." (".$e->getCode().")<br>\n";
}
echo "Doc recorded. id = ".$response->id." and revision = ".$response->rev."<br>\n";
// Doc recorded. id = 0162ff06747761f6d868c05b7aa8500f and revision = 1-249007504
```

Example : creating a document specifying the id

```
$new_doc = new stdClass();
$new_doc->title = "Some content";
$new_doc->_id = "BlogPost6576";
try {
    $response = $client->storeDoc($new_doc);
} catch (Exception $e) {
    echo "ERROR: ".$e->getMessage()." (".$e->getCode().")<br>\n";
}
echo "Doc recorded. id = ".$response->id." and revision = ".$response->rev."<br>\n";
// Doc recorded. id = BlogPost6576 and revision = 1-249004576
```

Example : updating an existing document :

```
// get the document
try {
    $doc = $client->getDoc('BlogPost6576');
} catch (Exception $e) {
    echo "ERROR: ".$e->getMessage()." (".$e->getCode().")<br>\n";
}

// make changes
$doc->title = 'Some smart content';
$doc->tags = array('twitter','facebook','msn');

// update the document on CouchDB server
try {
    $response = $client->storeDoc($doc);
} catch (Exception $e) {
    echo "ERROR: ".$e->getMessage()." (".$e->getCode().")<br>\n";
}
echo "Doc recorded. id = ".$response->id." and revision = ".$response->rev."<br>\n";
// Doc recorded. id = BlogPost6576 and revision = 2-456769086
```

Updating a document

Using CouchDB Update handlers , you can easily update any document part without having to send back the whole document.

updateDoc

PHPOnCouch\CouchClient::**updateDoc**(*$ddoc_id*, *$handler_name*, *$params*, *$doc_id = null*)

Update document according to the code defined in the update handler.

Params string $ddoc_id The desing document id (suffix of _design/)

Params string $handler_name The update handler name

Params array $params to complete...

Params string $doc_id The document id to udpate

Example : incrementing a document counter

Let's say we have a design document _design/myapp containing :

```
{
    "updates": {
        "bump-counter" : "function(doc, req) {
            if ( !doc ) return [null, {\"code\": 404, \"body\": \"Document not found / not s
            if (!doc.counter) doc.counter = 0;
            doc.counter += 1;
            var message = \"<h1>bumped it!</h1>\";
            return [doc, message];
        }"
    }
}
```

To bump the counter of the document "some_doc" , use :

```
$client->updateDoc("myapp","bump-counter",array(),"some_doc");
```

updateDocFullAPI

PHPOnCouch\CouchClient::**updateDocFullAPI**(*$ddoc_id*, *$handler_name*, *$options*)

Update document according to the code defined in the update handler.

params string $ddoc_id The id of the design document (suffix of _design/)

params string $handler_name Update handler name

params array $options An array of optionnal query modifiers :

- params : array|object of variable to pass in the URL (/?foo=bar)
- data : string|array|object data to set in the body of the request. If data is an array or an object it will be urlencoded using PHP http_build_query function and the request Content-Type header will be set to "application/x-www-form-urlencoded".
- Content-Type: string the Content-Type HTTP header to send to the couch server

Example :

```
$client->updateDocFullAPI("myapp","bump-counter",array( "data" => array("Something"=>"is set") )
```

deleteDoc

PHPOnCouch\CouchClient::**deleteDoc**(*$doc*)

Permanently removes $doc from the CouchDB server.

params stdClass $doc An object containing at least _id and _rev properties.

Example :

```
// get the document
try {
    $doc = $client->getDoc('BlogPost6576');
} catch (Exception $e) {
    echo "ERROR: ".$e->getMessage()." (".$e->getCode().")<br>\n";
}
// permanently remove the document
try {
    $client->deleteDoc($doc);
```

```
} catch (Exception $e) {
    echo "ERROR: ".$e->getMessage()." (".$e->getCode().")<br>\n";
}
```

copyDoc

PHPOnCouch\CouchClient::**copyDoc**(*$id*, *$new_id*)

Provides an handy way to copy a document.

params string $id The id of the document to copy.

params string $new_id The id of the new document.

returns The CouchDB server response, which has the main form than a document storage :

```
stdClass ( "ok" => true, "id" => "new_id" , "rev" => "1-23423423476" )
```

Example :

```
try {
    $response = $client->copyDoc('BlogPost6576','CopyOfBlogPost6576');
} catch (Exception $e) {
    echo "ERROR: ".$e->getMessage()." (".$e->getCode().")<br>\n";
}
```

Attachments

There is two methods handling attachments, it depends whether the file to send as attachment is on the harddrive, or if it's contained in a PHP variable. The first one should be more reliable for large attachments.

storeAttachment

PHPOnCouch\CouchClient::**storeAttachment**(*$doc*, *$file*, *$content_type = 'application/octet-stream'*, *$filename = null*)

Handles the process of storing an attachment on a CouchDB document.

params stdClass $doc A PHP object containing at least the properties _id and _rev

params string $file The complete path to the file on disk

params string $content_type The file's content-type

params string $filename The name of the attachment on CouchDB document, if the name is not the name of the file in $file

returns stdClass An HTTP response object like this :

```
stdClass ( "ok" => true, "id" => "BlogPost5676" , "rev" => "5-2342345476" )
```

Example :

```
$doc = $client->getDoc('BlogPost5676');
$ok = $client->storeAttachment($doc,'/etc/resolv.conf','text/plain', 'my-resolv.conf');
print_r($ok);
// stdClass ( "ok" => true, "id" => "BlogPost5676" , "rev" => "5-2342345476" )
```

storeAsAttachment

PHPOnCouch\CouchClient::**storeAsAttachment**(*$doc*, *$data*, *$filename*, *$content_type = 'application/octet-stream'*)

Records as a CouchDB document's attachment the content of a PHP variable.

params stdClass $doc A PHP object containing at least the properties _id and _rev

params mixed $data The data (the content of the attachment

params string $filename The name of the attachment on CouchDB document

params string $content_type The file's content-type

returns The HTTP response object.

Example :

```
$doc = $client->getDoc('BlogPost5676');
$google_home=file_get_contents('http://www.google.com/');
$ok = $client->storeAsAttachment($doc,$google_home,'GoogleHomepage.html','text/html');
print_r($ok);
// stdClass ( "ok" => true, "id" => "BlogPost5676" , "rev" => "5-2342345476" )
```

deleteAttachment

PHPOnCouch\CouchClient::**deleteAttachment**(*$doc*, *$attachment_name*)

Delete an attachment from a CouchDB document.

params stdClass $doc An object with, at least, _id and _rev properties,

params $attachment_name the name of the attachment to delete.

returns An stdClass representing the HTTP response.

Example :

```
$doc = $client->getDoc('BlogPost5676');
$ok = $client->deleteAttachment($doc,'GoogleHomepage.html');
```

getShow

PHPOnCouch\CouchClient::**getShow**(*$design_id*, *$name*, *$doc_id = null*, *$additionnal_parameters = array()*)

Request a show formatting of document *$doc_id* with show method *$name* stored in design document *design_id*.

Example :

```
$output = $client->getShow('blogs','html','BlogPost5676');
```

More infos on CouchDB show formatting here .

Bulk operations

A bulk operation is a unique query performing actions on several documents. CouchDB Bulk operations API are described in this wiki page .

keys

PHPOnCouch\CouchClient::**keys**(*$ids)->getAllDocs()*

To retrieve several documents in one go, knowing their IDs, select documents using the **keys($ids)** coupled with the method **getAllDocs()**. $ids is an array of documents IDs. This function acts like a view, so the output is the view output of CouchDB, and you should use "include_docs(true)" to have documents contents.

Example :

```
$view = $client->include_docs(true)->keys( array('BlogPost5676','BlogComments5676') )->getAllDoc
foreach ( $view->rows as $row ) {
  echo "doc id :".$row->doc->_id."\n";
}
```

storeDocs

PHPOnCouch\CouchClient::**storeDocs**(*$docs*, *$new_edits*)

To store several documents in one go, use the method **storeDocs($docs, $new_edits)**. $docs is an array containing the documents to store (as CouchDocuments, PHP stdClass or PHP arrays). $new_edits is related to the updates of the revision. If set to true (which is the default), assign new revision id for each update. When set to false, it prevents the database from assigning them new reivision IDS.

Example :

```
$docs = array (
    array('type'=>'blogpost','title'=>'post'),
    array('type'=>'blogcomment','blogpost'=>'post','depth'=>1),
    array('type'=>'blogcomment','blogpost'=>'post','depth'=>2)
);
$response = $client->storeDocs( $docs );
print_r($response);
```

which should give you something like :

```
Array
(
    [0] => stdClass Object
        (
            [id] => 8d7bebddc9828ed2edd052773968826b
            [rev] => 1-3988163576
        )

    [1] => stdClass Object
        (
            [id] => 37bcfd7d9e94c67617982527c67efe44
            [rev] => 1-1750264873
        )

    [2] => stdClass Object
```

```
            (
                [id] => 704a51a0b6448326152f8ffb8c3ea6be
                [rev] => 1-2477909627
            )

    )
```

This method also works to update documents.

deleteDocs

PHPOnCouch\CouchClient::**deleteDocs**(*$docs*, *$new_edits*)

To delete several documents in a single HTTP request, use the method **deleteDocs($docs, $new_edits)**. $docs is an array containing the documents to store (as couchDocuments, PHP stdClass or PHP arrays). $new_edits is related to the updates of the revision. If set to true (which is the default), assign new revision id for each update. When set to false, it prevents the database from assigning them new reivision IDS.

asArray

PHPOnCouch\CouchClient::**asArray**()

When converting a JSON object to PHP, we can choose the type of the value returned from a CouchClient query.

Take for example the following JSON object : { 'blog' : true, 'comments' : { 'title' : 'cool' } }

This can be converted into a PHP object :

```
stdClass Object
(
    [blog] => true
    [comments] => stdClass Object
        (
            [title] => "cool"
        )
)
```

OR into a PHP array :

```
Array
(
    [blog] => true
    [comments] => Array
        (
            [title] => "cool"
        )
)
```

Using the defaults, JSON objects are mapped to PHP objects. The **asArray()** method can be used to map JSON objects to PHP arrays.

Example:

```
$doc = $client->asArray()->getDoc('BlogPost5676');
print_r($doc);
```

should print :

```
Array (
    [id] => "BlogPost5676"
)
```

Mango Query

Summary

With the new release of CouchDB 2.0, Apache brought us the Mango Query. It's an adapted version of Cloudant Query for CouchDB. It's very similar to MongoDB Query syntax.

It lets you create indexes and perform queries with more ease that map/reduce. For more details, you may take a look at this :

- New feature: Mango Query
- Cloudant Query
- Mango source code

PHPOnCouch and Mango

With the recently added new available function to let you use the new Mango Query. This is very minimalist at the moment but feel free to suggest any ameliorations.

The Mango Query functionalities have been implemented directly in the CouchClient.

Functions

class PHPOnCouch**CouchClient**

getIndexes

PHPOnCouch\CouchClient**::getIndexes**()

This function returns you an array of indexes. Each index will have those properties :

•ddoc: The design document id of the index

•name: The name of the index

•type: The type of the index (special,text,json)

•def: The fields indexes

By default, you always have one index: _all_docs

Example :

```
$indexes = $client->getIndexes();

/*
[
    {
        "ddoc": null,
        "name": "_all_docs",
        "type": "special",
```

```
                "def": {
                    "fields": [
                        {
                            "_id": "asc"
                        }
                    ]
                }
            }
        ]
         */
        }
```

createIndex

PHPOnCouch\CouchClient::**createIndex**(*array $fields*, *$name = null*, *$ddoc = null*, *$type = 'json'*)

This function creates an index.

Params array $fields The fields that will be indexed

Params string $name The name of the index. If null, one will be generated by Couch.

Params string $ddoc The design document id to store the index. If null, CouchDB will create one.

Params string $type The type of the index. Only JSON is supported for the moment.

Returns stdClass An object. The object contains the following properties:

- result : Message that normally returns "created" or "exists"
- id : The id of the undex.
- name : The name of the index.

Example :

```
$index = client->createIndex(['firstName', 'birthDate', 'lastName'], 'personIdx', 'person');

/*
$index should give :
{
    "result":"created",
    "id":"_design/person",
    "name":"personIdx"
}
 */
```

find

PHPOnCouch\CouchClient::**find**(*$selector*, *$index = null*)

The new **find()** function let you query the database by using the new Mango Query. You can provide a selector query multiple fields and use conditional queries. You can sort your query and also determine which fields you want to retrieve. CouchDB will automatically select the most efficient index for your query but it's preferred to specify the index for faster results. Also, the **limit(number)** and **skip(number)** can be applied to the client before the query.

Supported query parameters

You can use the following query parameters :

•limit(number) : Limit the number of documents that will be returned.

•skip(n) : Skip n documents and return the documents following.

•sort(sortSyntax) : Array or values that follow the sort syntax

•fields(fieldsArray) : An array of fields that you want to return from the documents. If null, all the fields will be returned.

Params stdClass|array $selector A selector object or array that follows the Mango query documentation

Params string $index The name of the index to use("<design_document>" or ["<design_document>", "<index_name>"]). Otherwise automatically choosen.

Returns array Returns an array of documents

Example :

```
$selector = [
    '$and' =>
    [
        ['age' => ['$gt' => 16]],
        ['gender' => ['$eq' => 'Female']]
    ]
];
$docs = $client->skip(10)->limit(30)->sort(["age"])->fields(['firstName'])->find($selector);
```

explain

PHPOnCouch\CouchClient::**explain**(*$selector*, *$index = null*)

Let you perform a query like if you were using the *CouchClient::find* function. Therefore, the explain will not returns any documents. Instead, it will give you all the details about the query. For example, it could tell you which index has been automatically selected.

For the parameter, please refer to the *CouchClient::find* parameters.

Returns

It returns a object with a lot of detailed properties. Here are main properties :

- dbname : The name of the database
- index : Index object used to fullfil the query
- selector : The selector used for the query
- opts : The query options used for the query
- limit : The limit used
- skip : The skip parameter used
- fields : The fields returned by the query
- range : Range parameters passed to the underlying view

Example :

```
$selector = [
'year'=>['$gt'=>2010]
];
$details = $client->skip(0)->limit(2)->fields(['_id','_rev','year','title'])->sort(['year'=>
```

The $details values would be the equivalent in JSON :

```
{
    "dbname": "movies",
    "index": {
        "ddoc": "_design/0d61d9177426b1e2aa8d0fe732ec6e506f5d443c",
        "name": "0d61d9177426b1e2aa8d0fe732ec6e506f5d443c",
        "type": "json",
        "def": {
            "fields": [
                {
                    "year": "asc"
                }
            ]
        }
    },
    "selector": {
        "year": {
            "$gt": 2010
        }
    },
    "opts": {
        "use_index": [],
        "bookmark": "nil",
        "limit": 2,
        "skip": 0,
        "sort": {},
        "fields": [
            "_id",
            "_rev",
            "year",
            "title"
        ],
        "r": [
            49
        ],
        "conflicts": false
    },
    "limit": 2,
    "skip": 0,
    "fields": [
        "_id",
        "_rev",
        "year",
        "title"
    ],
    "range": {
        "start_key": [
            2010
        ],
        "end_key": [
            {}
        ]
```

```
            }
        }
```

Views

This section describes how to use PHP on Couch to retrieve views results from a CouchDB server.

Creating views

As said in the documentation , views are stored in CouchDB documents called *design documents*. So to create a view, you have to create a design document.

Example

```
$view_fn="function(doc) { emit(doc.timestamp,null); }";
$design_doc = new stdClass();
$design_doc->_id = '_design/all';
$design_doc->language = 'javascript';
$design_doc->views = array ( 'by_date'=> array ('map' => $view_fn ) );
$client->storeDoc($design_doc);
```

class PHPOnCouch**CouchClient**

getView

PHPOnCouch\CouchClient**::getView**(*$id*, *$name*)

The method **getView($id, $name)** sends back the CouchDB response of a view.

Params string $id is the design document id without '_design/'

Params string $name is the view name

Returns The view response object.

Example :

```
$result = $client->getView('all','by_date');
```

View response

The CouchDB response of a view is an object containing :

- **total_rows** , an integer of all documents available in the view, regardless of the query options
- **offset** , an integer givving the offset between the first row of the view and the first row contained in the resultset
- **rows** an array of objects.

Each object in **rows** contains the properties :

- **id** : the id of the emited document
- **key** : the emited key
- **value** : the emited value
- **doc** : the document object, if query parameter include_docs is set (read on for that).

Query parameters

The CoucClient implements chainable methods to add query parameters. The method names are mapped on their CouchDB counterparts :

- key
- keys
- startkey
- startkey_docid
- endkey
- endkey_docid
- limit
- stale
- descending
- skip
- group
- group_level
- reduce
- include_docs
- inclusive_end
- attachments

Example querying a view with a startkey, a limit and include_docs

```
$response = $client->startkey(100000000)->limit(100)->include_docs(true)->getView('all','by_date');
```

Which is the same as :

```
$client->startkey(100000000);
$client->limit(100);
$client->include_docs(true);
$response = $client->getView('all','by_date');
```

setQueryParameters

PHPOnCouch\CouchClient::**setQueryParameters**(*$params*)

You also can set query parameters with a PHP array, using the **setQueryParameters** method :

Params array $params A associative array of parameters to set.

Example:

```
$opts = array ( "include_docs" => true, "limit" => 10, "descending" => true );
$response = $client->setQueryParameters(opts)->getView("all","by_date");
```

asArray

PHPOnCouch\CouchClient::**asArray**()

When converting a JSON object to PHP, we can choose the type of the value returned from a CouchClient query.

Take for example the following JSON object :

```
{ "blog" : true, "comments" : { "title" : "cool" } }
```

This can be converted into a PHP object :

```
stdClass Object
(
    [blog] => true
    [comments] => stdClass Object
        (
            [title] => "cool"
        )
)
```

OR into a PHP array :

```
Array
(
    [blog] => true
    [comments] => Array
        (
            [title] => "cool"
        )
)
```

Using the defaults, JSON objects are mapped to PHP objects. The **asArray()** method can be used to map JSON objects to PHP arrays.

Example :

```
$response = $client->startkey(100000000)->limit(100)->include_docs(true)->asArray()->getView
```

Format a view with CouchDB list formatting feature

More infos on CouchDB lists .

getList

PHPOnCouch\CouchClient::**getList**(*$design_id*, *$name*, *$view_name*, *$additionnal_parameters = array()*)

This method retrieve a view and then format it using the algorithm of the $name list.

Params string $design_id The id of the design document(without the _design part)

Params string $name The name of the formatting algorithm.

Params string $view_name The name of the view to use.

Params array $additionnal_parameters The additionnal parameters.

Example :

```
$response = $client->limit(100)->include_docs(true)->getList('all','html','by_date');
// will run the view declared in _design/all and named *by_date*, and then
// pass it through the list declared in _design/all and named *html*.
```

getForeignList

PHPOnCouch\CouchClient::**getForeignList**(*$list_design_id*, *$name*, *$view_design_id*, *$view_name*, *$additionnal_parameters = array()*)

Retrieve a view defined in the document *_design/$view_design_id* and then format it using the algorithm of the list defined in the design document *_design/$list_design_id*.

Params string $list_design_id The list design id

Params string $view_design_id The view design id

Params array $additionnal_parameters The additionnal parameters that can be passed.

Example :

```
$response = $client->limit(100)->getForeignList('display','html','posts','by_date');
// will run the view declared in _design/posts and named *by_date*, and then
// pass it through the list declared in _design/display and named *html*.
```

getViewInfos

PHPOnCouch\CouchClient::**getViewInfos**(*$design_id*)

More info on view informations here

The method **getViewInfos($design_id)** sends back some useful informations about a particular design document.

Params string $design_id The id of the design document to use

Returns stdClass Returns an object with the following properties:

- name: The design document name
- view_index: View index informations

Example :

```
$response = $client->getViewInfos("mydesigndoc");
```

CouchAdmin class

Please read this first !!

The CouchAdmin class is only needed to **manage** users of a CouchDB server : add users, add admins, ...

You don't need the CouchAdmin class to connect to CouchDB with a login / password. You only need to add your login and password to the DSN argument when creating your CouchDB client :

```
$client = new CouchClient ("http://theuser:secretpass@couch.server.com:5984","mydatabase");
```

Managing CouchDB users

CouchDB rights management is really complex. This page can really help to understand how security is implemented in couchDB.

The **CouchAdmin** class contains helpful methods to create admins, users, and associate users to databases.

Synopsys

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
// Here my couchDB is in "admin party" mode (no user, no admin defined)
//
// I create an "anonymous" connector to the database
$client = new CouchClient ("http://localhost:5984/","mydb" );
// I then create an instance of the couchAdmin class, passing the couchClient as a parameter
$anonymous_adm = new CouchAdmin($client);

// I create the first admin user
try {
    $anonymous_adm->createAdmin("superAdmin","secretpass");
} catch ( Exception $e ) {
    die("unable to create admin user: ".$e->getMessage());
}

//
// now my database is not in "admin party" anymore.
// To continue Administration I need to setup an authenticated connector
//
$admclient = new CouchClient ("http://superAdmin:secretpass@localhost:5984/", "mydb" );
$adm = new CouchAdmin($admclient);

// create a regular (no superadmin) user)
try {
    $adm->createUser("joe","secret");
} catch ( Exception $e ) {
    die("unable to create regular user: ".$e->getMessage());
}

// set "joe" as admin of the database "mydb"
try {
    $adm->addDatabaseAdminUser("joe");
} catch ( Exception $e ) {
    die("unable to add joe to the admins list of mydb: ".$e->getMessage());
}

// Oh no I missed up remove "joe" from database "mydb" admins
try {
    $adm->removeDatabaseAdminUser("joe");
} catch ( Exception $e ) {
    die("unable to remove joe from the admins list of mydb: ".$e->getMessage());
}

// and add it to the members group of database "mydb"
try {
```

```
    $adm->addDatabaseMemberUser("joe");
} catch ( Exception $e ) {
    die("unable to add joe to the members list of mydb: ".$e->getMessage());
}

// well... get the list of users belonging to the "members" group of "mydb"
$users = $adm->getDatabaseMemberUsers();  // array ( "joe" )
```

Getting started

class PHPOnCouch**CouchAdmin**
The class that helps managing permissions, users and admins.

__construct

PHPOnCouch\CouchAdmin::**__construct** (*CouchClient $client*, *$options = array()*)
The CouchAdmin class constructor takes 2 parameters : a couchClient object and an array of configuration options.

Params CouchClient $client The CouchClient instance created with enough permissions to perform the administrative tasks.

Params array $options The options that can be passed to the CouchInstance and CouchAdmin. Here are the specific options for CouchAdmin :

- users_database : The user database to use (overwrite the default _users)
- node : The node to use for the configuration. **If it's not defined**, the first node of the *cluster_nodes* will be taken.

Example :

```
        // create a CouchClient instance
        $client = new CouchClient("http://localhost:5984/","mydb");
        // now create the CouchAdmin instance
        $adm = new CouchAdmin($client);
        // here $adm will connect to CouchDB without any credentials : that will only work if there
```

Admin party

On a fresh install, CouchDB is in **admin party** mode : that means any operation (create / delete databases, store documents and design documents) can be performed without any authentication.

Below is an example to configure the first server administrator, that we will name **couchAdmin** with the password **secretpass** :

```
// create an anonymous couchClient connection (no user/pass)
$client = new CouchClient("http://localhost:5984/","mydb");
// now create the couchAdmin instance
$adm = new CouchAdmin($client);
//create the server administrator
try {
    $adm->createAdmin("couchAdmin","secretpass");
} catch ( Exception $e ) {
    die ("Can't create server administrator : ".$e->getMessage());
}
```

Now that the couch server got a server administrator, it's not in "admin party" mode anymore : we can't create a second server administrator using the same, anonymous couchClient instance. We need to create a couchClient instance with the credentials of **couchAdmin**.

```
// create a server administrator couchClient connection
$client = new CouchClient("http://couchAdmin:secretpass@localhost:5984/","mydb");
// now create the CouchAdmin instance
$adm = new CouchAdmin($client);
```

Create users and admins

createAdmin

PHPOnCouch\CouchAdmin::**createAdmin**(*$login*, *$password*, *$roles = array()*)

Creates a CouchDB *server* administrator. A server administrator can do everything on a CouchDB server.

Params string $login The login of the new admin

Params string $password The raw password for the new admin.

Params array $roles The roles that will have this admin.

Example :

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

// Create an admin user
try {
    $adm->createAdmin("superAdmin","ommfgwtf");
} catch ( Exception $e ) {
    die("unable to create admin user: ".$e->getMessage());
}
```

createUser

PHPOnCouch\CouchAdmin::**createUser**(*$login*, *$password*, *$roles = array()*)

Creates a CouchDB user and returns it.

Params string $login The login of the new user

Params string $password The raw password for the new user.

Params array $roles The roles that will have this user.

Example :

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);
```

```
// Create a user
try {
    $adm->createUser("joe","dalton");
} catch ( Exception $e ) {
    die("unable to create user: ".$e->getMessage());
}
```

Example with roles

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

$roles = array ("thief","jailbreaker");

try {
    $adm->createUser("jack","dalton",$roles);
} catch ( Exception $e ) {
    die("unable to create user: ".$e->getMessage());
}
```

getUser

PHPOnCouch\CouchAdmin::**getUser**(*$login*)

The method returns the user document stored in the users database of the CouchDB server.

Params string $login The username of the user to find.

Returns The user if found. Otherwise, a CouchNotFoundException will be thrown.

Example :

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

// get a user
try {
    $joe = $adm->getUser("joe");
} catch ( Exception $e ) {
    if ( $e->getCode() == 404 ) {
        echo "User joe does not exist.";
    } else {
        die("unable to get user: ".$e->getMessage());
    }
}
```

getAllUsers

PHPOnCouch\CouchAdmin::**getAllUsers**()

The method returns the list of all users registered in the users database of the CouchDB server.

Note: This method calls a view, so you can use the view query options !

Returns An array of users found in the database.

Example :

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

// get all users
try {
    $all = $adm->getAllUsers();
} catch ( Exception $e ) {
    die("unable to get users: ".$e->getMessage());
}
print_r($all);

/** will print something like
Array (
    stdClass (
        "id" => "_design/_auth",
        "key" => "_design/_auth",
        "value" => stdClass (
                        "rev" => "1-54a591939c91922a35efee07eb2c3a72"
                    )
    ),
    stdClass (
        "id" => "org.couchdb.user:jack",
        "key" => "org.couchdb.user:jack",
        "value" => stdClass (
                        "rev" => "1-3e4dd4a7c5a9d422f8379f059fcfce98"
                    )
    ),
    stdClass (
        "id" => "org.couchdb.user:joe",
        "key" => "org.couchdb.user:joe",
        "value" => stdClass (
                        "rev" => "1-9456a56f060799567ec4560fccf34534"
                    )
    )
)
**/
```

Example - including user documents and not showing the design documents

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
```

```
        $all = $adm->include_docs(true)->startkey("org.couchdb.user:")->getAllUsers();
    } catch ( Exception $e ) {
        die("unable to get users: ".$e->getMessage());
    }
    print_r($all);

    /** will print something like
    Array (
        stdClass (
            "id" => "org.couchdb.user:jack",
            "key" => "org.couchdb.user:jack",
            "value" => stdClass (
                                "rev" => "1-3e4dd4a7c5a9d422f8379f059fcfce98"
                            ),
            "doc" => stdClass ( "_id" => "org.couchdb.user:jack", ... )
        ),
        stdClass (
            "id" => "org.couchdb.user:joe",
            "key" => "org.couchdb.user:joe",
            "value" => stdClass (
                                "rev" => "1-9456a56f060799567ec4560fccf34534"
                            ),
            "doc" => stdClass ( "_id" => "org.couchdb.user:joe", ... )
        )
    )
    **/
```

Removing users

Warning: This only works with CouchDB starting at version 1.0.1

deleteAdmin

PHPOnCouch\CouchAdmin::**deleteAdmin**(*$login*)

This permanently removes the admin $login.

Params string $login The username of the admin to delete.

Returns string Returns the hash of the password before it got removed.

Example : -hashed-0c796d26c439bec7445663c2c2a18933858a8fbb,f3ada55b560c7ca77e5a5cdf61d40e1a

Example : creating and immediately removing a server administrator

```
    <?php
    use PHPOnCouch\Couch,
        PHPOnCouch\CouchClient,
        PHPOnCouch\CouchAdmin;
    $client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
    $adm = new CouchAdmin($client);

    $adminLogin = "butterfly";
    $adminPass = "wing";
    try {
        $ok = $adm->createAdmin($adminLogin, $adminPass);
```

```
} catch (Exception $e) {
    die("unable to create admin user: ".$e->getMessage());
}
// here "butterfly" admin exists and can login to couchDB to manage the server

// now we remove it
try {
    $ok = $adm->deleteAdmin($adminLogin);
} catch (Exception $e) {
    die("unable to delete admin user: ".$e->getMessage());
}
// here "butterfly" admin does not exist anymore
```

deleteUser

PHPOnCouch\CouchAdmin::**deleteUser**(*$login*)

This method permanently removes the user $login.

Params string $login The login of the user to delete.

Example : removing a server user

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new couchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new couchAdmin($client);

try {
    $ok = $adm->deleteUser("joe");
} catch (Exception $e) {
    die("unable to delete user: ".$e->getMessage());
}
print_r($ok);

/** will print something like :
stdClass Object
(
    [ok] => 1
    [id] => org.couchdb.user:joe
    [rev] => 6-415784680cff486e2d0144ed39da2431
)
*/
```

Roles assignation

addRoleToUser

PHPOnCouch\CouchAdmin::**addRoleToUser**(*$user*, *$role*)

This method adds the role *$role* to the list of roles user *$user* belongs to. **$user** can be a PHP stdClass representing a CouchDB user object (as returned by getUser() method), or a user login.

Params string|stdClass $user The username of the user to edit or the User object returned by *CouchAdmin::getUser()* for example.

Params string $role The role to add to the specified user.

Example : adding the role *cowboy* to user *joe*

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $adm->addRoleToUser("joe","cowboy");
} catch ( Exception $e ) {
    die("unable to add a role to user: ".$e->getMessage());
}
echo "Joe now got role cowboy";
```

removeRoleFromUser

PHPOnCouch\CouchAdmin::**removeRoleFromUser**(*$user*, *$role*)

This method removes the role *$role* from the list of roles user *$user* belongs to. **$user** can be a PHP stdClass representing a CouchDB user object (as returned by getUser() method), or a user login.

Params string|stdClass $user The username of the user to edit or the User object returned by *CouchAdmin::getUser()* for example.

Params string $role The role to remove to the specified user.

Example : removing the role *cowboy* of user *joe*

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $adm->removeRoleFromUser("joe","cowboy");
} catch ( Exception $e ) {
    die("unable to remove a role of a user: ".$e->getMessage());
}
echo "Joe don't belongs to the cowboy role anymore";
```

setRolesToUser

PHPOnCouch\CouchAdmin::**setRolesToUser**(*$user*, *array $roles* =[ ])

This method let you set the roles for the selected user. A $user can either be the username of the user or a user object containing an **_id** and a **roles** property.

Example of usage :

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
```

```
$adm = new CouchAdmin($client);

try {
    $adm->setRolesForUser("joe",['tester','developer']);
    echo "Joe has now the tester and developer roles.";
} catch ( Exception $e ) {
    die("unable to remove a role of a user: ".$e->getMessage());
}
```

Database user security

CouchDB databases got two types of privileged users : the *members*, that can read all documents, and only write normal (non-design) documents. The *admins* got all privileges of the *members*, and they also can write design documents, use temporary views, add and remove *members* and *admins* of the database. The CouchDB wiki gives all details regarding rights management.

addDatabaseMemberUser

PHPOnCouch\CouchAdmin::**addDatabaseMemberUser**(*$login*)

This method adds a user in the members list of the database.

Params string $login The user to add to the member list of the current database

Example - adding joe to the members of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $adm->addDatabaseMemberUser("joe");
} catch ( Exception $e ) {
    die("unable to add user: ".$e->getMessage());
}
```

addDatabaseAdminUser

PHPOnCouch\CouchAdmin::**addDatabaseAdminUser**(*$login*)

Adds a user in the admins list of the database.

params string $login The user to add to the admin list of the current database

Example - adding joe to the admins of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $adm->addDatabaseAdminUser("joe");
```

```
} catch ( Exception $e ) {
    die("unable to add user: ".$e->getMessage());
}
```

getDatabaseMemberUsers

PHPOnCouch\CouchAdmin::**getDatabaseMemberUsers**()
Returns the list of users belonging to the *members* of the database.

Returns An array of usernames that belong to the member list of this database.

Example - getting all users beeing *members* of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $users = $adm->getDatabaseMemberUsers();
} catch ( Exception $e ) {
    die("unable to list users: ".$e->getMessage());
}
print_r($users);
// will echo something like: Array ( "joe" , "jack" )
```

getDatabaseAdminUsers

PHPOnCouch\CouchAdmin::**getDatabaseAdminUsers**()
Returns the list of users belonging to the *admins* of the database.

Returns An array of usernames that belong to the admin list of this database.

Example - getting all users beeing *admins* of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $users = $adm->getDatabaseAdminUsers();
} catch ( Exception $e ) {
    die("unable to list users: ".$e->getMessage());
}
print_r($users);
// will echo something like: Array ( "william" )
```

removeDatabaseMemberUser

PHPOnCouch\CouchAdmin::**removeDatabaseMemberUser**(*$login*)
Removes a user from the members list of the database.

Params string $login Remove the database username from the database member list.

Example - removing joe from the members of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $adm->removeDatabaseMemberUser("joe");
} catch ( Exception $e ) {
    die("unable to remove user: ".$e->getMessage());
}
```

removeDatabaseAdminUser

PHPOnCouch\CouchAdmin::**removeDatabaseAdminUser**(*$login*)

Removes a user from the admins list of the database.

Params string $login Remove the database username from the database admin list.

Example - removing joe from the admins of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $adm->removeDatabaseAdminUser("joe");
} catch ( Exception $e ) {
    die("unable to remove user: ".$e->getMessage());
}
```

Database roles security

Just like users, roles can be assigned as admins or members in a CouchDB database. The CouchDB wiki gives all details regarding rights management.

addDatabaseMemberRole

PHPOnCouch\CouchAdmin::**addDatabaseMemberRole**(*$role*)

Adds a role in the members list of the database.

Params string $role The role to add to the member role list of the current database.

Example - adding cowboy to the members of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
```

```
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $adm->addDatabaseMemberRole("cowboy");
} catch ( Exception $e ) {
    die("unable to add role: ".$e->getMessage());
}
```

addDatabaseAdminRole

PHPOnCouch\CouchAdmin::**addDatabaseAdminRole**(*$role*)

Adds a role in the admins list of the database.

Params string $role The role to add to the admin role list of the current database.

Example - adding *cowboy* role to the *admins* of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $adm->addDatabaseAdminrole("cowboy");
} catch ( Exception $e ) {
    die("unable to add role: ".$e->getMessage());
}
```

getDatabaseMemberRoles

PHPOnCouch\CouchAdmin::**getDatabaseMemberRoles**()

Returns the list of roles belonging to the *members* of the database.

Returns An array of roles belonging to the member section of the current database.

Example - getting all roles beeing *members* of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $roles = $adm->getDatabaseMemberRoles();
} catch ( Exception $e ) {
    die("unable to list roles: ".$e->getMessage());
}
print_r($roles);
// will echo something like: Array ( "cowboy" , "indians" )
```

getDatabaseAdminRoles

PHPOnCouch\CouchAdmin::**getDatabaseAdminRoles**()

Returns the list of roles belonging to the *admins* of the database.

Returns An array of roles belonging to the admin section of the current database.

Example - getting all roles beeing *admins* of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $roles = $adm->getDatabaseAdminRoles();
} catch ( Exception $e ) {
    die("unable to list roles: ".$e->getMessage());
}
print_r($roles);
// will echo something like: Array ( "martians" )
```

removeDatabaseMemberRole

PHPOnCouch\CouchAdmin::**removeDatabaseMemberRole**(*$role*)

Removes a role from the members list of the database.

Params string $role The role to remove from the database member role list.

Example - removing *cowboy* from the *members* of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $adm->removeDatabaseMemberRole("cowboy");
} catch ( Exception $e ) {
    die("unable to remove role: ".$e->getMessage());
}
```

removeDatabaseAdminRole

PHPOnCouch\CouchAdmin::**removeDatabaseAdminRole**(*$role*)

Removes a role from the admins list of the database.

Params string $role The role to remove from the database admin role list.

Example - removing *martians* from the admins of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
```

```
            PHPOnCouch\CouchAdmin;
        $client = new couchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
        $adm = new CouchAdmin($client);

        try {
            $adm->removeDatabaseAdminRole("martians");
        } catch ( Exception $e ) {
            die("unable to remove role: ".$e->getMessage());
        }
```

Accessing Database security object

Each Couch database got a security object. The security object is made like :

```
{
    "admins" : {
        "names" : ["joe", "phil"],
        "roles" : ["boss"]
    },
    "members" : {
        "names" : ["dave"],
        "roles" : ["producer", "consumer"]
    }
}
```

PHP on Couch provides methods to directly get and set the security object.

getSecurity

PHPOnCouch\CouchAdmin::**getSecurity**()

Returns Returns the security object of a CouchDB database.

Example - getting the security object of the database mydb

```
        <?php
        use PHPOnCouch\Couch,
            PHPOnCouch\CouchClient,
            PHPOnCouch\CouchAdmin;
        $client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
        $adm = new couchAdmin($client);

        try {
            $security = $adm->getSecurity();
        } catch ( Exception $e ) {
            die("unable to get security object: ".$e->getMessage());
        }
```

setSecurity

PHPOnCouch\CouchAdmin::**setSecurity**(*$security*)

Set the security object of a Couch database

Params stdClass $security The security object to set to the current database.

Example - setting the security object of the database mydb

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client);

try {
    $adm->setSecurity($security);
} catch ( Exception $e ) {
    die("unable to set security object: ".$e->getMessage());
}
```

setUserDatabase

PHPOnCouch\CouchAdmin::**setUserDatabase**(*$name*)

Set an alternate name for the users database on an already created couchAdmin instance.

Params string $name The name of the custom database to us to store users.

getUserDatabase

PHPOnCouch\CouchAdmin::**getUserDatabase**(*$name*)

Returns Return the name that is used actually to connect to the users database.

Database options

CouchDB got a special database used to store users. By default this database is called **_users**, but this can be changed.

CouchAdmin users_database

To create a CouchAdmin instance and specify the name of the users database, use the constructor second parameter $options, setting the option **users_database**:

Example - setting the couchdb users database name on couchAdmin object creation

```
<?php
use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin;
$client = new CouchClient ("http://couchAdmin:secretpass@localhost:5984/","mydb" );
$adm = new CouchAdmin($client, array ("users_database"=> "theUsers") );
```

You can also manipulate directly the CouchAdmin with the following methods : *CouchAdmin::setUserDatabase* and *CouchAdmin::getUserDatabase*.

CouchReplicator class

This section give details on using the CouchReplicator object.

Getting started

CouchDB supports replicating a database on other CouchDB databases. Think of replication as a copy-paste operation on databases.

The CouchReplicator object is a simple abstraction of the CouchDB replication model. Those replication features are available in CouchDB 0.11 . At the time of this coding, canceling a continuous replication doesn't seem to always work.

To create a new CouchReplicator object, you first have to include necessary files, and then instanciate the object, passing in argument a CouchClient instance.

```
<?php
 use PHPOnCouch\Couch,
    PHPOnCouch\CouchClient,
    PHPOnCouch\CouchAdmin,
    PHPOnCouch\CouchReplicator;

$client = new CouchClient ("http://localhost:5984/","mydb" );
// I create a replicator instance
$replicator = new CouchReplicator($client);
```

General functions

class PHPOnCouch**CouchReplicator**

to

PHPOnCouch\CouchReplicator::**to**()

To replicate a database to another existing database, use the **to**() method.

Example :

```
$client = new CouchClient ("http://localhost:5984/","mydb" );
// I create a replicator instance
$replicator = new CouchReplicator($client);
$response = $replicator->to("http://another.server.com:5984/mydb");
// database http://localhost:5984/mydb will be replicated to http://another.server.com:5984/
```

You can also replicato to a local database

```
$response = $replicator->to("mydb_backup");
// database http://localhost:5984/mydb will be replicated to http://localhost:5984/mydb_back
```

from

PHPOnCouch\CouchReplicator::**from**()

To replicate from a database to an existing database, use the **from**() method.

Example:

```
$response = $replicator->from("http://another.server.com:5984/mydb");
// database http://another.server.com:5984/mydb will be replicated to http://localhost:5984/
```

Note: Please note that CouchDB developpers hardly suggest to use the Pull replication mode : that means to prefer the "from()" method.

Chainable methods

create_target

PHPOnCouch\CouchReplicator::**create_target**()

The **create_target()** chainable method enables CouchDB to automatically create the target database, in case it doesn't exist.

Example :

```
$response = $replicator->create_target()->from("http://another.server.com:5984/mydb");
```

Which is equivalent to :

```
$replicator->create_target();
$response = $replicator->from("http://another.server.com:5984/mydb");
```

If the target database already exist, the create_target() method has no use.

doc_ids

PHPOnCouch\CouchReplicator::**doc_ids**()

To replicate only some documents, pass their ids to the **doc_ids()** chainable method.

Example :

```
$replicator->doc_ids( array ("some_doc", "some_other_doc") )->from("http://another.server.co
```

This code will replicate documents "some_doc" and "some_other_doc" of database "http://another.server.com:5984/mydb" to database "http://localhost:5984/mydb"

continous

PHPOnCouch\CouchReplicator::**continuous**()

A continuous replication is a replication that is permanent : once set, any change to the source database will be automatically propagated to the destination database. To setup a continuous replication, use the **continuous()** chainable method.

Example :

```
// setup a continuous replication
$replicator->continuous()->from("http://another.server.com:5984/mydb");
// create a CouchClient instance on the source database
$client2 = new CouchClient("http://another.server.com:5984/","mydb");
// create and record a document on the source database
$doc = new stdClass();
$doc->_id = "some_doc_on_another_server";
$doc->type = "foo";
$client2->storeDoc( $doc );
// let some time for CouchDB to replicate
sleep(10);
// read the document from the destination database
$doc = $client->getDoc("some_doc_on_another_server");
echo $doc->type;
```

cancel

PHPOnCouch\CouchReplicator::**cancel**()

To cancel a previously setup continuous replication, use the **cancel()** chainable method.

Example :

```
// setup a continuous replication
$replicator->continuous()->from("http://another.server.com:5984/mydb");
(...) //code code code
// remove the continuous replication
$replicator->cancel()->from("http://another.server.com:5984/mydb");
```

filter

PHPOnCouch\CouchReplicator::**filter**()

To have a full control over which document should be replicated, setup a filter definition on the source database. Then use the **filter()** chainable method to filter replicated documents.

```
// create a CouchClient instance pointing to the source database
$source_client = new CouchClient("http://localhost:5984","mydb");
// create a CouchClient instance pointing to the target database
$target_client = new CouchClient("http://another.server.com:5984","mydb")

// create a design doc
$doc = new stdClass();
$doc->_id = "_design/replication_rules";
$doc->language = "javascript";
// create a "no_design_doc" filter : only documents without the string "_design" will be rep
$doc->filters = array (
    "no_design_doc" => "function (doc, req) {
        if ( doc._id.match('_design') ) {
            return false;
        } else {
            return true;
        }
    }"
);
// store the design doc in the SOURCE database
$target_client->storeDoc($doc);

//create a CouchReplicator instance on the destination database
$replicator = new CouchReplicator($target_client);

// replicate source database to target database, using the "no_design_doc" filter
$replicator->filter('replication_rules/no_design_doc')->from($source_client->getDatabaseUri(
```

query_params

PHPOnCouch\CouchReplicator::**query_params**()

Filters can have a query parameters. This allows more generic filter codes. Let's modify the filter code above to pass the string to compare the document id to via query parameters :

```
// create a CouchClient instance pointing to the source database
$source_client = new CouchClient("http://localhost:5984","mydb");
// create a CouchClient instance pointing to the target database
```

```
        $target_client = new CouchClient("http://another.server.com:5984","mydb")

        // create a design doc
        $doc = new stdClass();
        $doc->_id = "_design/replication_rules";
        $doc->language = "javascript";
        // create a "no_design_doc" filter : only documents without the string "_design" will be rep
        $doc->filters = array (
            "no_str_in_doc" => "function (doc, req) {
                if ( doc._id.match( req.query.needle ) ) {
                    return false;
                } else {
                    return true;
                }
            }"
        );
        // store the design doc in the SOURCE database
        $target_client->storeDoc($doc);

        //create a CouchReplicator instance on the destination database
        $replicator = new CouchReplicator($target_client);

        // replicate source database to target database, using the "no_str_in_doc" filter, and setti
        $params = array ("needle"=>"_design");
        $replicator->query_params($params)->filter('replication_rules/no_str_in_doc')->from($source_
```

Replication of individual CouchDocuments

Please read the CouchDocument documentation to learn how to simply replicate a document to or from a database to another

CouchDocument class

This section give details on using CouchDocument data mapper.

CouchDocuments to simplify the code

CouchDB embed a simple JSON/REST HTTP API. You can simplify even more your PHP code using couch documents. Couch Documents take care of revision numbers, and automatically propagate updates on database.

Creating a new document

To create an empty CouchDocument, simply instanciate the **CouchDocument** class, passing the CouchClient object as the constructor argument.

Example :

```
$client = new CouchClient('http://localhost:5984/','myDB');
$doc = new CouchDocument($client);
```

If I set a property on $doc, it'll be registered in the database. If the property is not _id, the unique identifier will be automatically created by CouchDB, and available in the CouchDocument object.

Example :

```
$doc->type="contact";
echo $doc->id();
// 1961f10823408cc9e1cccc145d35d10d
```

However if you specify _id, that one will of course be used.

Example :

```
$doc = new CouchDocument($client);
$doc->_id = "some_doc";
echo $doc->id();
// some_doc
```

API Reference

class PHPOnCouch**CouchDocument**
A CouchDocument is a class that maps a document object.

set($key,$value)

PHPOnCouch\CouchDocument**::set**(*$key*, *$value = null*)
As we just saw, just set the property on the $doc object and it'll be recorded in the database. There are 2 ways to do it. You can either use the **set($key, $value)** method or simply use the setter **$obj->key = $value**.

Params string $key The key to set

Params string $value The value to set to the key.

Example :

```
$doc = new CouchDocument($client);
$doc->_id = "some_doc";
$doc->type = "page";
$doc->title = "Introduction";
```

set(array $params)

PHPOnCouch\CouchDocument**::set**(*array $params*)
It's always possible to set several properties in one query using the **set($params)** method

Params array $params An associative array of parameters that will be set.

Example using an array :

```
$doc = new CouchDocument($client);
$doc->set (
    array(
        '_id'   => 'some_doc',
        'type'  => "page",
        'title' => "Introduction"
    )
);
```

Example using an object

```
$prop = new stdClass();
$prop->_id = "some_doc";
$prop->type = "page";
$prop->title = "Introduction";

$doc = new CouchDocument($client);
$doc->set ( $prop );
```

setAutocommit

PHPOnCouch\CouchDocument::**setAutocommit**(*boolean $autoCommit*)

If, for some reason, you need to disable the auto-commit feature, use the **setAutocommit()** method. In this case, you'll have to explicitely call the **record()** method to store your changes on the database.

Params boolean $autoCommit Determine if the autocommit option should be enabled or not.

Example :

```
$doc = new CouchDocument($client);
$doc->setAutocommit(false);
$doc->_id = "some_doc";
$doc->type = "page";
$doc->title = "Introduction";
$doc->record();
```

record

PHPOnCouch\CouchDocument::**record**()

When the auto-commit feature is off, you need to apply changes manually. Calling the method **record()** apply the changes.

Example :

```
$doc = new CouchDocument($client);
$doc->setAutocommit(false);
$doc->_id = "some_doc";
$doc->type = "page";
$doc->title = "Introduction";
$doc->record();
```

getAutocommit

PHPOnCouch\CouchDocument::**getAutocommit**()

Returns True if autocommit is enabled. Otherwise false.

remove

PHPOnCouch\CouchDocument::**remove**(*$key*)

To unset a property, just use the **unset** PHP function, as you'll do for a PHP object. You can also use the **remove($key)** function which is normally called when you du a **unset**.

Params string $key The key of property to unset.

Example :

```
$prop = new stdClass();
$prop->_id = "some_doc";
$prop->type = "page";
$prop->title = "Introduction";

$doc = new CouchDocument($client);
$doc->set ( $prop );
unset($doc->title);
echo $doc->title ; // won't echo anything
```

getInstance

PHPOnCouch\CouchDocument::**getInstance**(*CouchClient $client*, *$docId*)

The static method **getInstance(CouchClient $client, $docId)** returns a CouchDocument when the specified id exists :

Params CouchClient $client The CouchClient instance initialized.

Params string $docId The _id of the document to use.

Example :

```
$doc = CouchDocument::getInstance($client,'some_doc');
echo $doc->_rev."\n";
echo $doc->type;
```

getUri

PHPOnCouch\CouchDocument::**getUri**()

The method **getUri()** sends back a string giving the current document URI.

Returns The document URI.

Example :

```
echo $doc->getUri();
/*
db.example.com:5984/testdb/dome_doc_id
*/
```

getFields

PHPOnCouch\CouchDocument::**getFields**()

To get the Couch document fields from a CouchDocument object, use the **getFields()** method

Returns Returns an object with the fields of the document.

Example :

```
$doc = CouchDocument::getInstance($client,'some_doc');
print_r($doc->getFields());
/*
    stdClass object {
        "_id"  => "some_doc",
        "_rev" => "3-234234255677684536",
```

```
                "type" => "page",
                "title"=> "Introduction"
            }
        */
```

storeAttachment

PHPOnCouch\CouchDocument::**storeAttachment**(*$file*, *$content_type = 'application/octet-stream'*, *$filename = null*)

Note: When the attachment is a file on-disk

Adds a new attachment or update the attachment if it already exists. The attachment contents is located on a file.

Params string $file The absolute path of the file.

Params string $content_type The Content-Type of the file.

Params string $filename The desired name of the stored attachment.

Example - Store the file /path/to/some/file.txt as an attachment of document id "some_doc" :

```
$doc = CouchDocument::getInstance($client,'some_doc');
try {
    $doc->storeAttachment("/path/to/some/file.txt","text/plain");
} catch (Exception $e) {
    echo "Error: attachment storage failed : ".$e->getMessage().' ('.$e->getCode().')';
}
```

storeAsAttachment

PHPOnCouch\CouchDocument::**storeAsAttachment**(*$data*, *$filename*, *$content_type = 'application/octet-stream'*)

Adds a new attachment, or update the attachment if it already exists. The attachment contents is contained in a PHP variable.

Params string $data The data to store as an attachment.

Params string $filename The desired name of the stored attachment.

Params string $content_type The Content-Type of the file.

Example - Store "Hello world !nAnother Line" as an attachment named "file.txt" on document "some_doc" :

```
$doc = CouchDocument::getInstance($client,'some_doc');
try {
    $doc->storeAsAttachment("Hello world !\nAnother Line", "file.txt" , "text/plain");
} catch (Exception $e) {
    echo "Error: attachment storage failed : ".$e->getMessage().' ('.$e->getCode().')';
}
```

deleteAttachment

PHPOnCouch\CouchDocument::**deleteAttachment**(*$name*)

Permanently removes an attachment from a document.

Params string $name The name of the attachment to delete.

Example - Deletes the attachment "file.txt" of document "some_doc" :

```
$doc = CouchDocument::getInstance($client,'some_doc');
try {
    $doc->deleteAttachment("file.txt");
} catch (Exception $e) {
    echo "Error: attachment removal failed : ".$e->getMessage().' ('.$e->getCode().')';
}
```

getAttachmentUri

PHPOnCouch\CouchDocument::**getAttachmentUri**(*$name*)

Params string $name The name of the attachment to get the URI.

Returns Returns the URI of an attachment.

Example :

```
$doc = CouchDocument::getInstance($client,'some_doc');
if ( $doc->_attachments ) {
    foreach ( $doc->_attachments as $name => $infos ) {
        echo $name.' '.$doc->getAttachmentURI($name);
        // should say something like "file.txt http://localhost:5984/dbname/some_doc/file.tx
    }
}
try {
    $doc->deleteAttachment("file.txt");
} catch (Exception $e) {
    echo "Error: attachment removal failed : ".$e->getMessage().' ('.$e->getCode().')';
}
```

replicateTo

PHPOnCouch\CouchDocument::**replicateTo**(*$url*, *$create_target = false*)

Replicate a CouchDocument to another CouchDB database. The create_target parameter let you create the remote database if it's not existing. The CouchDocuments instance provides an easy way to replicate a document to, or from, another database. Think about replication like a copy-paste operation of the document to CouchDB databases.

For those methods to work, you should have included the CouchReplicator class file src/CouchReplicator.php .

Params string $url The url of the remote database to replicate to.

Params boolean $create_target If true, create the target database if it doesn't exists.

Example :

```
$client = new CouchClient("http://couch.server.com:5984/","mydb");
// load an existing document
$doc = CouchDocument::getInstance($client,"some_doc_id");
```

```
// replicate document to another database
$doc->replicateTo("http://another.server.com:5984/mydb/");
```

replicateFrom

PHPOnCouch\CouchDocument::**replicateFrom**(*$id*, *$url*, *$create_target = false*)

Replicate a CouchDocument from another CouchDB database, and then load it into the CouchDocument instance.

Params string $id Replicate from this target document id.

Params string $url The url of the remote database to replicate to.

Params boolean $create_target If true, create the target database if it doesn't exists.

Example :

```
$client = new CouchClient("http://couch.server.com:5984/","mydb");
// load an existing document
$doc = new CouchDocument($client);

// replicate document from another database, and then load it into $doc
$doc->replicateFrom("some_doc_id","http://another.server.com:5984/mydb/");
echo $doc->_id ; (should return "some_doc_id")
$doc->type="foo"; // doc is recorded on "http://couch.server.com:5984/mydb"

// then replicate $doc back to http://another.server.com:5984/mydb/
$doc->replicateTo("http://another.server.com:5984/mydb/");
```

show

PHPOnCouch\CouchDocument::**show**(*$id*, *$name*, *$additionnal_parameters = array()*)

Parses the current document through a CouchDB show function.

Note: The show method is a proxy method to the **getShow()** method of **CouchClient**.

Params string $id The name of the _design document.

Params string $name The name of the show function

Params array $additionnal_parameters Additional parameters

Example : the database contains the following design document :

```
{
    "_id": "_design/clean",
    "shows": {
        "html": "function (doc, req) {
                    send('<p>ID: '+doc._id+', rev: '+doc._rev+'</p>');
                }"
    }
}
```

and another document that got the id "some_doc". We load the "some_doc" document as a CouchDocument object:

```
$doc = CouchDocument::getInstance($client,"some_doc");
```

We can then request CouchDB to parse this document through a show function :

```
$html = $doc->show("clean","html");
// html should contain "<p>ID: some_doc, rev: 3-2342342346</p>"
```

update

PHPOnCouch\CouchDocument**::update**(*$id*, *$name*, *$additionnal_params* = *array()*)

Allows to use the CouchDB update handlers feature to update an existing document. The CouchDocument object shouldd have an id for this to work ! Please see *CouchClient::updateDoc* method for more infos.

CHAPTER 4

Indices and tables

- genindex
- search
- modindex

p

Symbols

A

C

D

E

F

G

H

I

K

L

O

P

Q

R

S

T

U