OPEN POWER DRIVER

v 0.1 Manual

Tomasz Jędrzejewski
www.openpb.net

1. What is OPD?

Open Power Driver is open-source, written in **PHP 5.1**, modification of the original PHP Data Objects library. It has the same API, but adds some features that may be useful for many programmers:

1. Result caching
2. Main object factory
3. Query counter
4. Last query logger

It also modifies some aspects of OPD behaviour:

1. The default error reporting method is PDO::ERRMODE_EXCEPTION
2. The statement object is not an iterator (it will be fixed soon).

You may use the library in the same way, as the PDO library, so this manual describes only the differences. You should also know, how the PDO works in order to use OPD.

2. Installation and usage

In order to use OPD, you have to copy two files: *opd.class.php* and *opd.statement.php* from the */lib/* directory into your project's directory structure. Inside the application you define the *OPD_DIR* constant with the path to the library files and include *opd.class.php* file.

Your code should be enclosed inside a try...catch block, because OPD reports the problems as exceptions. You may initialize the main object manually (*opdClass* constructor with the same parameters, as the default PDO constructor uses) or use the *createOpd()* factory, which loads the configuration and configures the object. That's all.

Below, you may see a sample script that connects to the database.

```
<?php
     define('OPD_DIR', '../lib/');
     require(OPD_DIR.'opd.class.php');

     try
     {
          $sql = createOpd('./config.php');
          // your code goes here
     }
     catch(PDOException $exception)
     {
          opdErrorHandler($exception);
     }
?>
```

3. Open Power Driver reference

3.1 Classes and interfaces

The list of classes and interfaces included in OPD:

1. *opdClass* – the main class, replacement of the *PDO* class.
2. *iopdSatement* – the statement interface with the methods of the *PDOStatement* class.
3. *opdStatement* – default statement implementation.
4. *opdCachedStatement* – cached data statement.
5. *opdPreparedCacheStatement* – cached data from the *opdClass::prepare()* method statement.

OPD also makes use of these classes:

1. *Exception* – OPD errors and problems reported
2. *PDOException* – PDO errors and problems reported

3.2 opdClass methods

mixed opdClass::get(string $query);

The method executes specified query and returns the first column of the first row. It is useful, if we want to get some simple data, for example the total amount of products, because it allows to get it quickly without writing whole PHP code. If no rows returned, the method returns **NULL**.

void opdClass::setCacheDirectory(string $dir);

The method sets the directory, where the cached results are stored.

string opdClass::getCacheDirectory();

The method returns the current cache directory.

void opdClass::setCache(string $id [, bool $prepared]);

Enables the cache for the next query sent by:

1. *opdClass::query()* method, if the second parameter is not set.
2. *opdClass::prepare()* method, if the second parameter is set to *OPD_CACHE_PREPARE*

$id is the identifier of the cache file, where the results of this query are stored. Be careful, if your results depend on some external data, for example the category ID, be sure you have included this data into the cache ID. Otherwise there will be a conflict,

because different results use the same ID.

bool opdClass::clearCache(string $id);

Removes the cache file with the specified ID. You should always call this method, if you modify the content of a cached table. Otherwise the guests will not see the changes.

bool opdClass::clearCacheGroup(string $pattern);

Removes the cache files that match to the specified ID pattern. The pattern is parsed by PHP *glob()* function with *GLOB_BRACE* flag set.

int opdClass::getCounter();

Returns the number of queries executed since starting the connection.

3.3 OPD Functions and constants

opdClass createOpd(mixed $config);

OPD factory function. It returns the configured OPD object. *$config* may be both an array containing the configuration or the INI filename. The directives required by OPD are:

1. *dsn* – PDO dsn
2. *user* – connection user
3. *password* – connection password
4. *cache* – the cache directory (optional)

If the function fails, it generates an exception.

void opdErrorHandler(PDOException $exception);

Formats and shows the information about PDO error.

OPD_VERSION

Contains current OPD library version.

4. Result caching

The result caching is very simple in OPD, but it requires some additional code. Take a look at the example:

```
<?php
     define('OPD_DIR', '../lib/');
     require(OPD_DIR.'opd.class.php');

     try
     {
          $sql = createOpd('./config.php');

          $sql -> setCache('categories'); // 1
          $stmt = $sql -> query('SELECT id, name
               FROM categories ORDER BY name'); // 2
          $categories = array();
          while($row = $stmt -> fetch(PDO::FETCH_ASSOC)) // 3
          {
               echo $row['name'].'<br/>';
          }
          $stmt -> closeCursor(); // 4
     }
     catch(PDOException $exception)
     {
          opdErrorHandler($exception);
     }
?>
```

1. Here you tell the library you want to cache next query. Then, you specify the cache ID. OPD will save your results under this name. Notice that if your query contains some dynamic data, for example the category ID received from the guest, they must be also included in the cache ID. Otherwise OPD tries to save different results under the same name, which means trouble.
2. This is the query we want to send. If the data are already cached, this query is not physically sent to the database, but the data are loaded from a file.
3. Returning the data.
4. Calling this method is necessary both in PDO and OPD. It saves the cached

result onto the HDD.

You may also cache the queries executed with the *prepare()* method. Take a look at the pseudocode:

```
$sql -> setCache('exec_1', OPD_CACHE_PREPARE);
$sql -> setCache('exec_2', OPD_CACHE_PREPARE);
// ...
$sql -> setCache('exec_n', OPD_CACHE_PREPARE);
$stmt = $sql -> prepare('query');
//...
$stmt -> execute(); // 1st time
$stmt -> execute(); // 2nd time
//...
$stmt -> execute(); // n-th time
```

Before you call *prepare()* method, you have to set the cache IDs for all bindings and executions you want to do. If you would like to avoid the caching for any of them, remaining it for others, specify the boolean „false” as the cache ID and this statement execution will not be cached at all.

Additional calls of the *execute()* methods give no effect, if there was no cache ID defined for them.

Notice that if you modify the data in the tables with cached content, you have to remove the cache files manually with *setCache()* or *setCacheGroup()* methods. Caching for the specified time peroid, which does not need such practises, will be implemented in the next version of the OPD.

5. To Do

The list of features that will be implemented in future versions:

1. Caching of PDO::FETCH_CLASS and PDO::FETCH_OBJECT data. Currently the behaviour is unknown and you should not use them with the caching.
2. Implementing the *Iterator* interface by the statement classes.
3. Caching of *iopdStatement::fetchColumn()* data.
4. Caching the results for the specified time peroid.

6. Authors and support

Open Power Driver library is a part of Open Power Board project. It is written by Tomasz Jędrzejewski (Poland). If you have any suggestions or support requests, please visit our discussion board: www.openpb.net/forum. If you found any bug, visit our bugtracker: www.openpb.net/bugs.

Thank you for choosing Open Power Driver. Try also our template engine: Open Power Template (opt.openpb.net)!