TorneLIB\Module\Config\WrapperConfig

WrapperConfig takes care of most configuration in the netcurl module. It also handles exceptions that should be thrown back to developers/clients/users.

Options

Settings that will be read by the curldriver (and the other drivers) is stored in WrapperConfig-options. They are reachable with the following method calls:

Method	Arguments/Return	Description
setOptions	array	Replaces internal options with new defined options.
getOptions	array	Returns the current options array.
setOption	$key, $value	Inserts a new option in the current option.
getOption	$key	Returns the setting of the current option, or throws an error if it's not set.
deleteOption	$key	Removes an option key that is (eventually) set.
replaceOption	$key, $value, $oldKey	Replaces an old option key with a new. *Example: This is internally used to switch between timeouts set in seconds and milliseconds.*

Magics and timeouts

WrapperConfig is built on a magics base. By means, WrapperConfig can preconfigure data that is not normally supported by the wrapper. Eeven if it is currently mostly untested, you can simply use either setOption to prepare data that should be passed through to curl, or your own set-call. For example using code that looks like this:

```
$config = (new WrapperConfig())->setProxy('proxy-data');
$curlWrap = (new CurlWrapper())->setConfig($config);
```

```
$wrapper = new CurlWrapper();
$currentConfig = $wrapper->getConfig();
$currentConfig->setProxy('proxy-data');
$curlWrap = $wrapper->setConfig($currentConfig);
```

However, while this library is being built, proxy- and ip configuration will be a part of WrapperConfig. The magics idea will however keep the codebase and "IDE helpers" quite clean. An already implemented example is the timeout. See below in the supported magics section.

Supported Magics

Method	Parameters	Description

setTimeout	$timeout = integer $useMillisec = boolean	Set up connectivity timeouts. For curl, there are two values being handled in this method: connection timeouts and timeouts for the entire request. The $timeout variable sets up the timeout to a number of seconds or milliseconds for how long the entire request session should be. The connect timeout will be the half of this value. Using the second argument as a boolean, you can decide if the timeout should be set in either seconds or milliseconds. The curloptions (in this case) will replace itself with either of the values. The method getTimeout returns information about the configured values in an arrayed format: `return [ 'CONNECT' => connect_timeout_integer, 'REQUEST' => entire_timeout_integer, 'MILLISEC' => is_integer_or_not_integer_boolean, ];`

Authentication and HTTP-data

WrapperConfig is handling data storing of every http-request being made with netcurl. Each module built should use this configuration wrapper to set up and prepare for the request. In MODULE_CURL (6.0) a setAuthentication-method was stored internally and then shared to each module that was linked to MODULE_CURL. For example, setAuthentication in that module was inherited to the Soap module. For 6.1, authentication data should be fetched separately from each module. The curl module has its own authentication method, but it uses WrapperConfig to store it. In this way, other modules can reach the same data. Other data that is also stored in this wrapper can be seen below.

Variable	Reached by methods	Description
requestDataType	setRequestDataType getRequestDataType	Stores request data type (TorneLIB\Model\Type\dataType) that defines how the request will look when post data is included. **Example: JSON, SOAP, XML, etc**
requestData	setRequestData getRequestData	Stores the request post-data in a proper format. For example, if the request if based on JSON, the postdata is transformed to a json object. If it is a post, it most likely will look like ***variable=value&secondVariable=secondValue***, etc
requestMethod	setRequestMethod getRequestMethod	GET, POST, DELETE, PUT, etc
authData	setAuthentication getAuthentication	Stores authenication data for the request. authTypes is based on TorneLIB\Model\Type\authType (which is a curl transformed list of constants).

Throwables

Throwables in WrapperConfig is based on web traffic and returned headers. In the default state, everything that returns HTTP 400 - 599 are considered throwables and will throw an exception if returned from a http request. There is a slight differense here though, since netcurl 6.1 uses an improved errorhandler. For example, the curlwrapper no longer needs to be instatiated in the same way as 6.0 required to extract exceptions from the responses. For 6.0, you had to do something like this:

```
$instance = new MODULE_CURL();
try {
    $request = $instance->doGet('url');
} catch (Exception $e) {
    // Extract responses from the primary instance as $request will not be available during an exception.
    $instance->getBody();
}
```

Instead the errorhandler makes this possible (taken from the unittest **getThrowablesByBody**):

```
use TorneLIB\Exception\ExceptionHandler;

try {
    // A shorter example of a request that returns a body with json-content.
    $curlWrapper = new CurlWrapper()->request('url');
} catch (ExceptionHandler $e) {
    $curlWrapperException = $e->getExtendException()->getParsed();
}
```

Exceptions look

When exceptions are passing through the ExceptionHandler, exceptions are described more detailed compared to v6.0. The error message now not only contains the error string, it also contains - if it is returned from the server - the header error message too. The error example below is based on the **getThrowablesByBody** above. And yes, even if it returns an error in the header the extendedException is a copy of the CurlWrapper in this case, so it can extract furthermore data from the body.

```
Error 404 returned from server: "404".
```

Since the message body has a json-object that contains another message, we can also extract that directly from the exception via the getParsed() method:

```
Not found
```

The example is based on a document request, where document is missing, rather than the URL itself.