The Network and cURL class (tornevall_network.php)

Live information

There's a Mailinglist put up for everything concerning netcurl. That's also where you can find release information (for now). You can subscribe to the list here
.

DEPRECATED

This document is deprecated. Take a look at v6.1 instead.

This page

This documentation is about to get outdated. Things here is under maintenance only.

Going further down will give you deprecated documentation.

Join the project

Jump to the project tracker to join this project - or report bugs!

Source code can be viewed at https://bitbucket.tornevall.net/projects/LIB/repos/tornelib-php/browse/classes/tornevall_network.php

Inline docs for the develop-version is also located at http://phpdoc.tornevall.net/TorneLIBv5/class-TorneLIB.Tornevall_cURL.html

Why rebuilding the wheel? There are other libraries that do the same job!

Well. Yes. Almost. The other libraries out there, are probably doing the exact same job as we do here. The **problem** with other libraries (that I've found) is amongst others that they are way too big. Taking for example GuzzleHTTP, is for example a huge project if you're aiming to use a smaller project that not requires tons of files to run. They probably covers a bit more solutions than this project, however, we are aiming to make curl usable on as many places as possible in a smaller format. What we are doing here is turning the curl PHP libraries into a very verbose state and with that, returning completed and parsed data to your PHP applications in a way where you don't have to think of this yourself.

So, this library is built for doing one thing only: Communicate. So instead of including one big package of library-files, this library shoud probably be considered a light weight curl-handler. You could of course use other libraries aswell, but our goal is to keep this one as small as possible, just to be able to fit anywhere you'd like. The whole idea with this curl bridge is to avoid fiddling with all curl settings by yourself.

History

People sometimes wonder why we are (probably) rebuilding the wheel as others have done this too before, but this library are born from a very old project, that was built for testing proxies scraped from different kind of sites and lists. The first step that was made then, was to build a client that actually could scrape sites without having to set up a whole bunch of scripts and libraries (this happened around 2000-2006 so, in that moment, we did know very little about which libraries that existed). As the source code probably reveals it has a quite big section, just to configure tunnels and proxies in different methods. And this is also why this library exists.

As we was scraping proxies, we also needed to find out how the proxy acted, so there is plenty of time wasted on this. Sometimes for example, when connecting to a socks proxy, we not only flagged the proxy as working. We continued to compare the connection to the outbound connection and if the ip address on the outgoing interfaces was matching the "inbound". If we found a mismatch, this proxy became flagged as an "elite proxy" (which this behaviour is called). We also wanted to scan for proxies which revealed the origin ip by HTTP_VIA and similar headers. In short, there's a whole bunch of scenarios, that we could not get out from a library unless we built it ourselves. So, here we are!

Dependencies

To get the most out of this library, there is currently one dependency to php-xml which makes the SOAP client work. Normally, when starting on an empty machine (Ubuntu), very little things needs to be done to run:

Dependencies
`apt-get install php-curl php-xml`

PHP-cURL simplifier library

The network- and cURL class is an independent class library that handles network related things. The cURL library especially, has special features, like parsing data received from the body. For example, calling an URL that returns json strings, you'll get an object back that you could handle immediately, instead of parsing the data yourself.

Instantiation is being done by running a simple $CURL = new \TorneLIB\Tornevall_cURL();

If the curl library is missing (we are checking that curl_init() exists) an exception will be thrown as the instantiation is in progress. In early versions of the library an exception are thrown as soon as the library file gets loaded, which might stop a site completely.

The cURL class

We are with the cURL class in a quite verbose mode, to get the most out of the web request you are doing. The response are separated in an array as follows

array key	**array content value**
code	The current HTTP Status code returned from the request (See https://en.wikipedia.org/wiki/List_of_HTTP_status_codes) This response status code can be used to detect, for example 404 not found errors, or similar, of a page. By means, you may throw optional exceptions by checking this code state as you need.

header	The full response header returned from the request as ...
	
	Header array
	`array( 'info' => array(), 'full' => 'Full header as string' );`
	
	... where **info** is an array with keys and values. Example:
	
	InfoArray
	`array( "Server" => "Apache", "Strict-Transport-Security" => "max-age=63072000; includeSubDomains", "X-Frame-Options" => "SAMEORIGIN", "X-Powered-By" => "PHP/5.6.27" );`

body	The full returned body of the request
parsed	If the body are recognized as specially formatted (json, xml, etc), the parsed array will transform into an object or an array, depending on the content. Currently, the parser supports simpler modes like XML (SimpleXMLElement) and by this also RSS-feeds (unconfirmed), JSON and serialized data. The curl library also supports usage of the PEAR-package XML_Serializer.

Posting parameters

As of the release of **20161213** (which can be found in the repo), all functions that allows postData to be sent in a web call may be posted quite freely. You can either put postData in the standard format like:

`&param1=value1&param2=value2`

You may also send your requested data through the function as an array() or even a JSON-object. Such call may look like this:

```
doPost("http://test.com/", $postDataObject, \TorneLIB\CURL_POST_AS::POST_AS_JSON)
// Alternative without the POST_AS-constant
// doPost("http://test.com/", $postDataObject, 1)
```

The function itself tries to detect if the input $postDataObject is encoded or decoded (*keep in mind that problems with NAN-values or similar is not supported*).

Simple usage example

There are four basic function calls, that by default handles http calls automatically. Normally, no further settings are required except for calling an URL. Posting data could be made manually or as an object. It also handles proxies, outgoing ip-addresses and hopefully also the ability to switch to explicit ipv6- or ipv4-mode.

A very basic call to the library may look like this. By using the "parsed" key in the response, you'll get any supported output formatted correctly.

Basic cURL call

```
require_once("tornevall_network.php");
$CURL = new \TorneLIB\Tornevall_cURL();
$getResponse = $CURL->doGet("https://api.tornevall.net/2.0/endpoints/getEndPoints");
print_r($getResponse['parsed']);
```

For the moment the library supports following formats:

- PHP serialized objects
- json encoded data

- xml data
  By their tags xml a SimpleXMLElement-decode will invoke as the response is received, which gives us support for both rss amongst others.

Note: SOAP is using our class Tornevall_SimpleSoap to handle soap-calls - by adding ?wsdl, &wsdl (depending on how the post/get-parameters looks) or setting CURL_POST_AS to POST_AS_SOAP, the SOAP handler will be activated - or not.

Making SOAP calls

Example:

SOAP Example

```
require_once("tornevall_network.php");
$CURL = new \TorneLIB\Tornevall_cURL();
$Response = $CURL->doGet("https://test.com/wsdlService?wsdl");
echo "<pre>";
try {
    $getResponse = $Response->getOneResponseHere();
    print_R($getResponse['parsed']);
} catch (Exception $e) {
    echo "Exception Thrown: " . $e->getMessage() . "\n";
    $getResponse = $Response->getLibResponse();
}
echo "</pre>";
```

The doGet()-parameter may also look like as of the version released 20161216:

```
$Response = $CURL->doGet("https://test.com/requestService?wsdl", \TorneLIB\CURL_POST_AS::POST_AS_SOAP);
```

Alternative, without the POST_AS-constant:

```
$Response = $CURL->doGet("https://test.com/requestService?wsdl", 2);
```

SSL Certificates and verification

The cUrl library has it's own method to find out if there are missing SSL certificates during https-calls. Normally, this is not a very big issue, since standard linux installations are smart enough to store bundles, pems and crt-files in standard locations (/etc/ssl/certs for example). This is however not always the case. If the storage is located elsewhere, https-calls may sometimes fail. By using $CURLsslPemLocations, you can replace the current standard paths and let the library look for them where you believe they should be instead, since this can not be set on fly through ini_set. This whole operation may be tested through the call TestCerts(), that runs an internal function openssl_guess(). This function is always called from the primary constructor, to make sure the certificates can be located as early as possible and it a certificate file are found in the inital run, it does not have to be runned again. Later on, when using the primary GET/POST/etc-calls the certificate bundle will be used in the stream_context-handler to make sure the https gets handled properly.

This solution helps server admins with restricted access to files on disk, so if the certificates are not available, this could be fixed with sslPemLocations, like this:

sslPemLocations

```
require_once("tornevall_network.php");
$CURL = new \TorneLIB\Tornevall_cURL();
$CURL->sslPemLocations = array('/home/users/myTestUserName/certs/sslBundles.pem');
$CURL->doGet("https://my-test-url.com");
```

Peer and host verification

Since version 5.0.0-20170210 (LIB-59 - Getting issue details... STATUS) the peer and host verification has been hardened, so if a system runs an insecure configuration (where SSL certificate bundles are missing), disabling the peer/host verification must actively be done at initialization of the library, as in the example below. However, if your system contains SSL certificates, but located somewhere else than in a default location, this could be configured by the above example with sslPemLocations.

However, from PHP 5.6.0 this should normally not be necessary, since openssl_get_cert_locations() should normally take care of this, through the local function of the library - **openssl_guess()**.

Disable peer verification example

```
require_once("tornevall_network.php");
$CURL = new \TorneLIB\Tornevall_cURL();
$CURL->setSslUnverified(true);
$output = $CURL->doGet("https://my-test-url.com");
```

SSL Capability checking

In some release packages, where curl is not compiled with SSL support, the library might spit out a lot of exceptions. With 5.0.0-20170425, the library will come with the SSL capability check. With this, a developer might discover before any problem occurs, if it is because of missing SSL libraries in curl. The function call is very easy to use: *$LIB->hasSsl()* returns a boolean of the first SSL control check, that is being made when the library is about to instantiate.

Parsing regular HTML

The library has, as of 5.0.0-20120211, the ability to convert regular incoming html-documents into a parsed array, if there is a need to read any content as an array. Since this ability may consume memory and/or CPU, it has to be enabled first. It has been tested with HTML documents, so far and it extracts a document by following content:

- **ByNodes**
  Extracted in the simplest way, by nodes, meaning every each of the elements are sorted out with information about the elements tagnames and attributes (name and id)
- **ByClosestTag**
  Extracts the DOMDocument and sets its closest element identification to each of the array variables, beginning with the tagnameelement name element id.
- **ById**
  Extracts the DOMDocument and sets the element identification by its id (getElementById equivalent)

HTML Content Parsing

```
require_once("tornevall_network.php");
$CURL = new \TorneLIB\Tornevall_cURL();
$CURL->setParseHtml(true);
$output = $CURL->doGet("https://my-test-url.com");
var_dump($output['parsed']['ById']);
```