Module: NETCURL

Content of this page

Other documents

Exceptions handling

Compatibility span (Supported PHP versions)

This library *should* be compatible with at least PHP 5.3 up to PHP 7.2. However, running this module in older versions of PHP makes it more unreliable as PHP is continuosly developed. There are no guarantees that the module is fully functional from obsolete releases (like PHP 5.3 and most of the PHP 5.x-series is).

Requirements and dependencies

Some kind of a supported driver is needed. NetCURL was built for curl so it could be a great idea to have curl available in your system. The goal with the module is however to pick up the best available driver in your system.

Supported drivers

- curl
- soap (SoapClient with XML)
- Guzzle
- Wordpress

Future plans for independence:

- Streams
- Sockets

Dependencies and installation

As netcurl is built to be independently running dependencies is not necessesary required. To reach full functionality the list below might be good to have available.

- **Installation:** Composer. NetCURL can be bundled/manually downloaded, but the best practice is to install via composer. Otherwise you're a bit on your own.
- **SSL (OpenSSL):** Not having SSL available means that you won't be able to speak https

- **SOAP:** To make use of the SOAP components in NetCurl, **XML** libraries and **SoapClient** needs to be there. SoapClient uses Streams to fetch wsdl.

XML, CURL, SOAP

In apt-based systems, extra libraries can be installed with commands such as:

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

The module installation itself

This is the recommended way (and only officially supported) of installing the package.

1. Get composer.
2. Run composer:

```
composer require tornevall/tornelib-php-netcurl
```

Composer also adds the crypto module during the installation. API-docs can be found here, as this site is not yet updated with this.

NetCurl Parsing

- By PHP serialized content (serialize/unserialize)
- JSON
- XML (Starting with SimpleXMLElements, the PEAR/PECL modules such as XmlSerializer/XMLUnserializer are also supported)
- DOMDocument (Converts html documents into DOMDocument arrays, by three methods: **nodes**, **closest tags** and **element ids**)

NetCurl extra features

The name NetCurl was born from the fact that there are more than one module in this class file. A network module also resides in the single file, which is the primary reason why it is called **Net**Curl. NetCurl takes care of different network based features, like *resolving hosts, making sure that ip-addresses is really ip-addresses, extracts proxy addresses from server variables, encodes base64-strings into URL-compatible strings, reads CIDR-formatted netmasks properly*, etc, etc, etc. This part of the documentation has documentation of all those modules.

Internal drivers

Driver	Class	Provides	Installation	Notes
CURL	MODULE_CURL	***curl***	Curl must be installed	
SimpleSoap	MODULE_SOAP	***SoapClient*** *(Streams)*		

Importable drivers

NetCurl primarily supports curl, as the name reveals. However, if you run wordpress or Guzzle, there is a small driver switch in NetCurl that makes it possible to survive without curl. Below, the supported drivers are listed and what they may provide for you.

Driver	Class	Provides	Installation	Notes
Wordpress	`WP_HTTP`	***curl*** and **streams**	WP must be present	On SOAP, SoapClient makes fallback to internal class
GuzzleHttp	`GuzzleHttp\Client` `GuzzleHttp\Handler\StreamHandler`	***curl*** and **streams**	Add to composer.json: "guzzlehttp/guzzle": "6.3.0"	On SOAP, SoapClient makes fallback to internal class

Driver overriders

Overriding the default drivers can be done by running for example.

Say that you'd prefer to use Guzzle over CURL. First of all, this is kind of double building the network module as guzzle primarily uses CURL to run. So doing this will make you build a wrapper over a wrapper, instead of going directly through curl.

First of all, you need to get the guzzle package. This is done by adding this row into your composer.json-file (or use the composer require statement).

```
"guzzlehttp/guzzle": "^6.3"
```

When this is done, use composer install or composer update and start adding your code:

```
$CURL = new MODULE_CURL();
$CURL->setDriver(NETCURL_NETWORK_DRIVERS::DRIVER_GUZZLEHTTP);
// or
$CURL->setDriver(NETCURL_NETWORK_DRIVERS::DRIVER_GUZZLEHTTP_STREAM);
```

The best available driver could also be autodetected via the method **setDriverAuto()**.

Library Content

NetCurl contains a bunch of classes in a single file that in the beginning was supposed to make the library independent. The namespace is TorneLIB.

Class	Description	API Document
MODULE_NETWORK	Handler of any networking related actions	Link
MODULE_CURL	The magic class that does the work for you in all http-related actions	Link
MODULE_SOAP	SoapClient-handler	Link
NETCURL_EXCEPTIONS	Exceptions related to the library	Link
NETCURL_AUTH_TYPES	Available authentication types for use with password protected sites	Link
NETCURL_POST_METHODS	List of methods available in this library (Like GET, POST, PUT, DELETE)	Link
NETCURL_POST_DATATYPES	Prepared formatting for POST-content in this library (Also available from for example PUT) Examples: json, xml, etc	Link
NETCURL_RESOLVER	Resolver methods that is available when trying to connect (ipv4/ipv6 based selectors). This defines whether the resolver should prioritize ipv4 or ipv6 first.	Link
NETCURL_NETWORK_DRIVERS	Available http driver handlers - Internal (default), WordPress, Guzzle, etc	Link
~~NETCURL_ENVIRONMENT~~	Used for testing (**deprecated**)	Link
NETCURL_RESPONSETYPE	Responsetypes (how they are returned - as arrays or objects) *Normally not used*	Link
NETCURL_HTTP_OBJECT	If you need to get the responses out as objects, this is the predefined object	Link
NETCURL_IP_PROTOCOLS	Address Types class (ipv4/ipv6) This defined wheter the connector should prioritize ipv4 or ipv6 first	Link
MODULE_NETBITS	Bitmasking handler (will probably be moved out of this library soon)	Link

NetCurl responses (Unchained)

The list below shows the response content that are returned from a GET/POST/etc. You necessarily do not need to fetch the keys yourself; you can also use NetCurl internal methods to get each data key from the response. As of v6.0.18, NetCurl is chained by default. This means that chained (PHP 5.4+) commands can be used (like **$LIB->doGet()->getParsed()**). Chaining can be disabled with ***$LIB->setChain(false);*** - however, chaining is automatically disabled if PHP 5.3 or lower are detected. The table below shows how the raw data content looks "unchained".

Response key	Get method	array content value
code	**getCode()**	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	**getBody()**	The full returned body of the request
parsed	**getParsed()**	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.