License

This software is licensed under the GNU Lesser General Public License v3.0.

Changelog

Version 3.1.0

- Tested with Zabbix 6.0.
- Added missing namespace in examples.
- Added function `loginToken()` to log in via API tokens.

Version 3.0.2

- Fixed bug in getApiVersion() with invalid credentials.
- Added example and documentation using composer. See `examples/composer`.

Version 3.0.1

- Tested with Zabbix 5.0 - 5.4.
- Added support for Zabbix 5.4 API change in `user.login` method.
- Added namespace `IntelliTrend\Zabbix`.
- Added composer support.
- Moved to semver version numbers.

No version change

- Added more examples to filter by hostnames, hostids, hostgroupnames and hostgroupids.

Version 2.8

- Tested with Zabbix 5.0 and 5.2.
- Ensure that params passed to API call are an array.
- Added library version to debug output.

Version 2.7

- BREAKING CHANGE: Classfilename renamed from `Zabbixapi.php` to `ZabbixApi.php` to match classname.
- Call to `getAuthKey()` no longer simply returns the `authKey`. If there was no previous call to the Zabbix-API this funcion will call the Zabbix-API to ensure a valid key before returning the key.
- Fixed error message for invalid sslCaFile.

Version 2.6

- Public release.
- Added check for Curl.
- Added example for filtering and additional params passed to the Zabbix-API.
- Call to `login()` no longer initially calls the Zabbix-API anylonger to verify the `authKey`.

- If debug is enabled via option, or via function before calling `login()`, `login()` issues a call to the Zabbix-API to check wether the session is re-used.

Version 2.5

- Internal release.

Version 2.4

- Initial public release.

Zabbix API PHP Client with session caching and SSL support

There are quite a lot of Zabbix API clients out that work really well with Zabbix. However most of them do not support `session caching` or need tweaks to work with self signed certificates.

This library aims to solve those problems. It supports the following features:

- Session caching.
- HTTPS connections with official- and self-signed certificates.
- Works with Linux and Windows PHP implementations.
- Multiple concurrent connections with different user accounts and/or different servers.
- Zabbix versions: 3.0, 3.2, 3.4, 4.0, 4.2, 4.4, 5.0, 5.2, 5.4 and 6.0.
- No installation required.

It is commercially backed up and maintained by IntelliTrend GmbH, an official Zabbix Partner and Zabbix Training company.

There was also a lightning talk Zabbix API – the easy way about using this library on the Zabbix conference 2020 in Benelux.

Why Session caching?

How authentication via API usally works

Each time an application uses `user.login`, the Zabbix API creates a so called `AuthKey`. (See user.login). This key is then passed via each request through the `auth` request property.

In most cases, the API libary does this transparently for the user. However, if the script is called later on again (i.e by a cron job), a new `user.login` is performed, thus creating a new `AuthKey` for that new session.

Zabbix keeps those session, until a session expired or is logged out. You can check your actual settings through the Zabbix frontend: `Administration/General/Housekeeper - Section Sessions`. The default value is 365days.

This means, any created session (if there is no logout) will be kept for 365 days.

Assume we have a set of 10 API scripts that run every hour. This means we will create `10 x 24 = 240` sessions per day. Using more scripts or a smaller interval will of cause increase this number.

Note: Zabbix 5.4 and later support API auth tokens, which won't create sessions when used. Therefore, no session management is required on the client side, either.

The problem with many sessions in the session table

Everytime a request hits the Zabbix frontend, either per webbrowser or as a JSON RPC API request, Zabbix has to verify the request against the existing sessions. The more sessions to verify, the longer this will take.

> **Note:** We have seen installations with millions of sessions, where frontend access slowed down considerable by 15sec+ per request. Keep in mind that for example the dashboard not only performs one request, but multiple requests depending on the number of widgets used.

So the best is to `reuse` a session, until it expires. This is where session caching steps in.

Using session caching with the Zabbix API

When a new session is created, the AuthKey is saved encrypted to disk. This is similar to using a cookie in a classic webbrowser. If a new request is performed, the existing session is read once and the AuthKey is reused.

So to follow up the example given before: Calling a script using session-caching, even over a month, will create just `1` session.

If the AuthKey becomes invalid, the libary automatically performs a new `user.login`, re-executes the failed request and updates the stored session. All of this happens in the background, the user of the libary has not to deal with it.

How does session encryption work, what about multiple sessions?

Each session has a unique name based on a hash using the `zabbixUserName` and the `zabbixUrl`. The session itself is encrypted using the `zabbixUserName` and the `zabbixPassword`.

This way, the libary can be used with different useraccounts and also different zabbix server instances.

Where are sessions stored?

Sessions are stored by default in the users `tmp` directory. However there is a config option `sessionDir` that allows to override this setting. See the detailed description below.

Installation

> **Note:** The PHP environment **must have CURL** installed. `ZabbixApi.php` has a built-in check for curl and will throw an exception if curl is missing.

Without using composer

There is no installation required. Simply copy the file `ZabbixApi.php` and use the class.

```
require_once "ZabbixApi.php";

use IntelliTrend\Zabbix\ZabbixApi;

$zbx = new ZabbixApi();
```

Using composer

```
require('vendor/autoload.php');

use IntelliTrend\Zabbix\ZabbixApi;

$zbx = new ZabbixApi();
```

See `examples/composer` for a full example with details.

Usage

Error handling

The library makes use of `Exceptions`. There is no need to check each response value. Simply wrap the calls in `try/catch blocks`.

Depending on where an error occurs, different error codes are passed to the exception object, together with the message property.

- `Zabbix API errors`: The original API error code and message is passed.
- `Connection and SSL errors`: The original CURL error code and message is passed.
- `Library errors`: A constant error code, as defined in the class constant EXCEPTION_CLASS_CODE, and a useful message is passed. Default=1000.

Configuration

The class is configured when calling the `login` method. Any further Zabbix API call is performed by the `call` method.

> **Note:** One can run multiple instances at the same time, connecting with different user accounts to the same zabbix server or to another zabbix server.

Basic usage

Lets start with a very simple example:

```
require_once "ZabbixApi.php";
use IntelliTrend\Zabbix\ZabbixApi;
$zbx = new ZabbixApi();
try {
    $zbx->login('https://my.zabbixurl.com/zabbix', 'myusername', 'mypassword');
    $result = $zbx->call('host.get', array("countOutput" => true));
    print "Number of hosts:$result\n";
} catch (Exception $e) {
    print "==== Exception ===\n";
    print 'Errorcode: '.$e->getCode()."\n";
    print 'ErrorMessage: '.$e->getMessage()."\n";
    exit;
}
```

Basically this is all needed. The `call` method is transparent to the Zabbix API definition. It takes 2 parameter: `$method` and `$params` as specified for the particular Zabbix API method.

Instead of a login with a user name and password, you can also use an API token in Zabbix 5.4+:

```
require_once "ZabbixApi.php";
use IntelliTrend\Zabbix\ZabbixApi;
$zbx = new ZabbixApi();
try {
    $zbx->loginToken('https://my.zabbixurl.com/zabbix',
'123456789abcdef123456789abcdef123456789abcdef123456789abcdef1234');
    $result = $zbx->call('host.get', array("countOutput" => true));
    print "Number of hosts:$result\n";
} catch (Exception $e) {
    print "==== Exception ===\n";
    print 'Errorcode: '.$e->getCode()."\n";
    print 'ErrorMessage: '.$e->getMessage()."\n";
    exit;
}
```

For example to retrieve all 'Host Groups' with all properties, we can do this:

```
require_once "ZabbixApi.php";
use IntelliTrend\Zabbix\ZabbixApi;
$zbx = new ZabbixApi();
$zabUrl = 'https://my.zabbixurl.com/zabbix';
$zabUser = 'myusername';
$zabPassword = 'mypassword';
try {
    $zbx->login($zabUrl, $zabUser, $zabPassword);
    $result = $zbx->call('hostgroup.get', array("output" => 'extend'));
    foreach ($result as $hostGroup) {
        $hostGroupId = $hostGroup['groupid'];
        $hostGroupName = $hostGroup['name'];
        print "groupid:$hostGroupId, hostGroupName:$hostGroupName\n";
    }
} catch (Exception $e) {
    print "==== Exception ===\n";
    print 'Errorcode: '.$e->getCode()."\n";
    print 'ErrorMessage: '.$e->getMessage()."\n";
    exit;
}
```

> **Note:** This second example would not create a new session when calling `login` again after the first example. It would reuse the session from the previous example. `login` returns true when an existing session was found.

Advanced usage and SSL Options

The basic example works fine, even with HTTPS, given there is a valid certificate the php installation is aware of. But what todo when using self-signed certificates?

Here we can use the optional `options` argument when calling `login` to setup the SSL options.

Example - Turn off SSL verification:

```
require_once "ZabbixApi.php";
use IntelliTrend\Zabbix\ZabbixApi;
$zbx = new ZabbixApi();
```

```
$options = array('sslVerifyPeer' => false, 'sslVerifyHost' => false);
try {
    $zbx->login('https://my.zabbixurl.com/zabbix', 'myusername', 'mypassword',
$options);
    $result = $zbx->call('host.get', array("countOutput" => true));
    print "Number of hosts:$result\n";
} catch (Exception $e) {
    print "==== Exception ===\n";
    print 'Errorcode: '.$e->getCode()."\n";
    print 'ErrorMessage: '.$e->getMessage()."\n";
    exit;
}
```

Using filters and field selectors

It is quite easy to pass filter and field selectors through the API. Basically any params defined by the Zabbix-API can be passed this way.

Example - Select first 5 hosts filtered by status, maintenance_status and type and add their groups and macros.

```
require_once "ZabbixApi.php";
use IntelliTrend\Zabbix\ZabbixApi;
$zbx = new ZabbixApi();
try {
    // default is to verify certificate and hostname
    $options = array('sslVerifyPeer' => false, 'sslVerifyHost' => false);
    $zbx->login('https://my.zabbixurl.com/zabbix', 'myusername', 'mypassword',
$options);

    print
"================================================================================\n";
    // Get hosts and other information available to this useraccount, but
filtered and limited
    $limit = 5;
    $params = array(
        'output' => array('hostid', 'host', 'name', 'status',
'maintenance_status', 'description'),
        'filter' => array('status' => 0, 'maintenance_status' => 0, 'type' =>
1),
        'selectGroups' => array('groupid', 'name'),
        'selectInterfaces' => array('interfaceid', 'main', 'type', 'useip',
'ip', 'dns', 'port'),
        'selectInventory' => array('os', 'contact', 'location'),
        'selectMacros' => array('macro', 'value'),
        'limit' => $limit
    );

    $result = $zbx->call('host.get',$params);
    print "==== Filtered hostlist with groups and macros ====\n";
    foreach($result as $host) {
        printf("HostId:%d - Host:%s\n", $host['hostid'], $host['host']);
        foreach($host['groups'] as $group) {
            printf("    - GroupId:%d - Group:%s\n", $group['groupid'],
$group['name']);
        }
        foreach($host['macros'] as $macro) {
```

```
            printf("    - Macro:%s - Value:%s\n", $macro['macro'],
$macro['value']);
        }

    }

} catch (Exception $e) {
    print "==== Exception ===\n";
    print 'Errorcode: '.$e->getCode()."\n";
    print 'ErrorMessage: '.$e->getMessage()."\n";
    exit;
}
```

Debug Mode

The class provides a debug mode that ouputs a lot of details. To enable, either use an option or function.

Debug Option as param in login():

```
$zbx = new ZabbixApi();
try {
    $options = array('sslVerifyPeer' => false, 'sslVerifyHost' => false, 'debug'
=> true);
    $zbx->login('https://my.zabbixurl.com/zabbix', 'myusername', 'mypassword',
$options);
    ...
```

Debug Function - can be used any time:

```
# Turn debug on
$zbx = new ZabbixApi();
$zbx->setDebug(true);
try {
    $options = array('sslVerifyPeer' => false, 'sslVerifyHost' => false);
    $zbx->login('https://my.zabbixurl.com/zabbix', 'myusername', 'mypassword',
$options);
    ...
    # Can be turned off anytime
    $zbx->setDebug(false);
    ...
    # And turned on again
    $zbx->setDebug(true);
```

Functions reference

Basic functions

login($zabUrl, $zabUser, $zabPassword, $options)

Initial login. Configures the class and loads a cached session if it exists. It does not executes a request to the remote server to test the credentials or session at this time. This happens automatically during the first call to the Zabbix-API. One can enforce the validation by calling `getAuthKey()` after `login()`.

> **Note:** If debug is enabled via option, or via function before calling `login()`, `login()` issues a call to the Zabbix-API to check wether the session is re-used.

- `return` void
- `throws` Exception $e. Invalid options, session issues or connection problems.
- `param` string $zabUrl
- `param` string $zabUser
- `param` string $zabPassword
- `param` array $options - optional settings.
  Example: `array('sslVerifyPeer' => false, 'sslVerifyHost' => false);`
  - `debug`: boolean - default=false. Show debug information. Also setDebug() can be used. Default is false
  - `sessionDir`: string - default=user `tmp` directory. Directory where to store the sessions.
  - `sslCaFile`: string - default=use php.ini settings. Filename of external CACertBundle. Useful when using self signed certificates from internal CA. See the CURL or Mozilla websites for those bundles.
  - `sslVerifyPeer`: boolean - default=true. Verify certificate. Throws Exception on failure. When false, ignore any verfication errors.
  - `sslVerifyHost`: boolean - default=true. Verify Hostname against CN in certificate. Only works if certificate can be validated.

    > **Note:** If sslVerifyPeer=false but the certificate itself is valid and the hostname does not match, then sslVerifyHost=true will raise an exception.

  - `useGzip`: boolean - default=true. Use gzip compression for requests.
  - `connectTimeout`: integer - default=10. Max. time in seconds to connect to server.
  - `timeout`: default=30. Max. time in seconds to process request.

loginToken($zabUrl, $zabToken, $options)

Alternative login method using an API token, which is set directly as auth token. Options are the same as for `login()`.

Since session management is not required with API tokens, no cached sessions are loaded or saved.

- `return` void
- `throws` Exception $e. Invalid options, session issues or connection problems.
- `param` string $zabUrl
- `param` string $zabToken
- `param` array $options - optional settings.

call($method, $params)

Execute Zabbix API call. Will automatically login/re-login and retry if the call failed using the current authKey read from session.

> **Note:** Can only be called after login() was called once before at any time.

- `return` mixed $reusedSession. Decoded Json response from API call or a scalar. See Zabbix API documentation for details.
- `throws` Exception $e. API Error, Session issues or connection problems.

- `param` string $method. Zabbix API method i.e. 'host.get'
- `param` mixed $params. Params as defined in the Zabbix API for that particular method.

logout()

Logout from Zabbix Server and also delete the authKey from filesystem.

> **Note:** Only use this method if its really needed, because the session cannot be reused later on. Logouts on API object created with an API token have no effect.

- `return` void
- `throws` Exception $e. API Error, Session issues or connection problems

setDebug($state)

Enable / Disable debug output. Can be used any time.

- `return` void
- `param` boolean $state. True enables debug output.

Convenient functions

getVersion()

Get version of this library.

- `return` string $version.

getApiVersion()

Get Zabbix API version from Zabbix Server. Also useful to check wether the Zabbix Api url is valid.

> **Note:** Prefer this function over `call('apiinfo.version')`, because it does not try to authenticate and stores the Zabbix Api version for further requests.

- `return` string $version. Uses API method 'apiinfo.version'.
- `throws` Exception $e. API Error, Session issues or connection problems

Utility functions

getAuthKey()

Get authKey used for API communication.

> **Note:** If there was no previous call to the Zabbix-API, this funcion will call the Zabbix-API to ensure a valid authkey.

- `return` string $authKey.

getSessionDir()

Get session directory.

- `return` string $directory.

getSessionFileName()

Get session FileName storing the encrypted authKey without path.

- `return` string $fileName.

getSessionFile()

Get full FileName with path.

- `return` string $fileName.

Tips and Tricks

Common "get" method parameters

As described in the Zabbix-Api, there several parameter that can be added to a "get" method.

Find attached a list of some less known but quite useful ones, especially `preservekeys` :

Parameter	Type	Description
countOutput	boolean	Return the number of records in the result instead of the actual data.
editable	boolean	If set to true return only objects that the user has write permissions to. Default: false.
limit	integer	Limit the number of records returned.
preservekeys	boolean	Use IDs as keys in the resulting array.