API PHP class

This is PHP class for working with RouterOS v3 API. You can take it, edit it and use it as you need.

NOTE - The class as shown does not work for large replies

Author

Denis Basta (Denis [dot] Basta [at] gmail [dot] com)

Contributors

Nick Barnes
Ben Menking (ben [at] infotechsc [dot] com)
Jeremy Jefferson (http://jeremyj.com)
Cristian Deluxe (djcristiandeluxe [at] gmail [dot] com)

Changelog

1.0 **Denis Basta** (Denis [dot] Basta [at] gmail [dot] com) First PHP Class released from author

1.1 **Nick Barnes**
read() function altered to take into account the placing of the "!done" reply and also correct calculation of the reply length.

1.2 **Ben Menking** (ben [at] infotechsc [dot] com)
read() function altered removed echo statement that dumped byte data to screen

1.3 **Jeremy Jefferson** (http://jeremyj.com)
January 8, 2010
Fixed write function in order to allow for queries to be executed

1.4 **Cristian Deluxe** (djcristiandeluxe [at] gmail [dot] com)
November 17, 2011
comm() function altered, added the possibility of make regular exp queries.
parse_response() and parse_response4smarty() functions altered to support a "single data" responses from server
Added documentation to functions following PHPDoc guidelines
Added version number (1.4) for follow the changes more easy

Class

```
<?php

/******************************

 *

 * RouterOS PHP API class v1.4

 * Author: Denis Basta

 * Contributors:

 *    Nick Barnes

 *    Ben Menking (ben [at] infotechsc [dot] com)

 *    Jeremy Jefferson (http://jeremyj.com)

 *    Cristian Deluxe (djcristiandeluxe [at] gmail [dot] com)

 *

 * http://www.mikrotik.com
```

```
 * http://wiki.mikrotik.com/wiki/API_PHP_class

 *

 ******************************/


class routeros_api

{

    var $debug = false;      // Show debug information

    var $error_no;           // Variable for storing connection error number, if any

    var $error_str;          // Variable for storing connection error text, if any

    var $attempts = 5;       // Connection attempt count

    var $connected = false;  // Connection state

    var $delay = 3;          // Delay between connection attempts in seconds

    var $port = 8728;        // Port to connect to

    var $timeout = 3;        // Connection attempt timeout and data read timeout

    var $socket;             // Variable for storing socket resource


    /**

     * Print text for debug purposes

     *

     * @param string      $text       Text to print

     *

     * @return void

     */

    function debug($text)

    {

        if ($this->debug)

            echo $text . "\n";

    }




    /**

     *

     *

     * @param string        $length

     *

     * @return void

     */

    function encode_length($length)

    {

        if ($length < 0x80) {

            $length = chr($length);

        } else if ($length < 0x4000) {

            $length |= 0x8000;

            $length = chr(($length >> 8) & 0xFF) . chr($length & 0xFF);

        } else if ($length < 0x200000) {

            $length |= 0xC00000;

            $length = chr(($length >> 16) & 0xFF) . chr(($length >> 8) & 0xFF) . chr($length & 0xFF);
```

```
        } else if ($length < 0x10000000) {
            $length |= 0xE0000000;
            $length = chr(($length >> 24) & 0xFF) . chr(($length >> 16) & 0xFF) . chr(($length >> 8) & 0xFF) . chr($length & 0xFF);
        } else if ($length >= 0x10000000)
            $length = chr(0xF0) . chr(($length >> 24) & 0xFF) . chr(($length >> 16) & 0xFF) . chr(($length >> 8) & 0xFF) . chr($length & 0xFF);
        return $length;
    }


    /**
     * Login to RouterOS
     *
     * @param string      $ip         Hostname (IP or domain) of the RouterOS server
     * @param string      $login      The RouterOS username
     * @param string      $password   The RouterOS password
     *
     * @return boolean                If we are connected or not
     */
    function connect($ip, $login, $password)
    {
        for ($ATTEMPT = 1; $ATTEMPT <= $this->attempts; $ATTEMPT++) {
            $this->connected = false;
            $this->debug('Connection attempt #' . $ATTEMPT . ' to ' . $ip . ':' . $this->port . '...');
            if ($this->socket = @fsockopen($ip, $this->port, $this->error_no, $this->error_str, $this->timeout)) {
                socket_set_timeout($this->socket, $this->timeout);
                $this->write('/login');
                $RESPONSE = $this->read(false);
                if ($RESPONSE[0] == '!done') {
                    if (preg_match_all('/[^=]+/i', $RESPONSE[1], $MATCHES)) {
                        if ($MATCHES[0][0] == 'ret' && strlen($MATCHES[0][1]) == 32) {
                            $this->write('/login', false);
                            $this->write('=name=' . $login, false);
                            $this->write('=response=00' . md5(chr(0) . $password . pack('H*', $MATCHES[0][1])));
                            $RESPONSE = $this->read(false);
                            if ($RESPONSE[0] == '!done') {
                                $this->connected = true;
                                break;
                            }
                        }
                    }
                }
                fclose($this->socket);
            }
            sleep($this->delay);
        }
        if ($this->connected)
            $this->debug('Connected...');
```

```
        else
            $this->debug('Error...');
        return $this->connected;
    }



    /**
     * Disconnect from RouterOS
     *
     * @return void
     */
    function disconnect()
    {
        fclose($this->socket);
        $this->connected = false;
        $this->debug('Disconnected...');
    }



    /**
     * Parse response from Router OS
     *
     * @param array       $response   Response data
     *
     * @return array                  Array with parsed data
     */
    function parse_response($response)
    {
        if (is_array($response)) {
            $PARSED      = array();
            $CURRENT     = null;
            $singlevalue = null;
            $count       = 0;
            foreach ($response as $x) {
                if (in_array($x, array(
                    '!fatal',
                    '!re',
                    '!trap'
                ))) {
                    if ($x == '!re') {
                        $CURRENT =& $PARSED[];
                    } else
                        $CURRENT =& $PARSED[$x][];
                } else if ($x != '!done') {
                    if (preg_match_all('/[^=]+/i', $x, $MATCHES)) {
                        if ($MATCHES[0][0] == 'ret') {
                            $singlevalue = $MATCHES[0][1];
```

```
                    }
                                $CURRENT[$MATCHES[0][0]] = (isset($MATCHES[0][1]) ? $MATCHES[0][1] : '');
                            }
            }
        }
        if (empty($PARSED) && !is_null($singlevalue)) {
            $PARSED = $singlevalue;
        }
        return $PARSED;
    } else
        return array();
}



/**
 * Parse response from Router OS
 *
 * @param array       $response   Response data
 *
 * @return array                  Array with parsed data
 */
function parse_response4smarty($response)
{
    if (is_array($response)) {
        $PARSED  = array();
        $CURRENT = null;
        $singlevalue = null;
        foreach ($response as $x) {
            if (in_array($x, array(
                '!fatal',
                '!re',
                '!trap'
            ))) {
                if ($x == '!re')
                    $CURRENT =& $PARSED[];
                else
                    $CURRENT =& $PARSED[$x][];
            } else if ($x != '!done') {
                if (preg_match_all('/[^=]+/i', $x, $MATCHES)) {
                    if ($MATCHES[0][0] == 'ret') {
                        $singlevalue = $MATCHES[0][1];
                    }
                    $CURRENT[$MATCHES[0][0]] = (isset($MATCHES[0][1]) ? $MATCHES[0][1] : '');
                          }
            }
        }
        foreach ($PARSED as $key => $value) {
```

```
            $PARSED[$key] = $this->array_change_key_name($value);
        }
        return $PARSED;
        if (empty($PARSED) && !is_null($singlevalue)) {
            $PARSED = $singlevalue;
        }
    } else {
        return array();
    }
}



/**
 * Change "-" and "/" from array key to "_"
 *
 * @param array       $array      Input array
 *
 * @return array                  Array with changed key names
 */
function array_change_key_name(&$array)
{
    if (is_array($array)) {
        foreach ($array as $k => $v) {
            $tmp = str_replace("-", "_", $k);
            $tmp = str_replace("/", "_", $tmp);
            if ($tmp) {
                $array_new[$tmp] = $v;
            } else {
                $array_new[$k] = $v;
            }
        }
        return $array_new;
    } else {
        return $array;
    }
}



/**
 * Read data from Router OS
 *
 * @param boolean     $parse      Parse the data? default: true
 *
 * @return array                  Array with parsed or unparsed data
 */
function read($parse = true)
{
```

```
        $RESPONSE = array();
        while (true) {
            // Read the first byte of input which gives us some or all of the length
            // of the remaining reply.
            $BYTE   = ord(fread($this->socket, 1));
            $LENGTH = 0;
            // If the first bit is set then we need to remove the first four bits, shift left 8
            // and then read another byte in.
            // We repeat this for the second and third bits.
            // If the fourth bit is set, we need to remove anything left in the first byte
            // and then read in yet another byte.
            if ($BYTE & 128) {
                if (($BYTE & 192) == 128) {
                    $LENGTH = (($BYTE & 63) << 8) + ord(fread($this->socket, 1));
                } else {
                    if (($BYTE & 224) == 192) {
                        $LENGTH = (($BYTE & 31) << 8) + ord(fread($this->socket, 1));
                        $LENGTH = ($LENGTH << 8) + ord(fread($this->socket, 1));
                    } else {
                        if (($BYTE & 240) == 224) {
                            $LENGTH = (($BYTE & 15) << 8) + ord(fread($this->socket, 1));
                            $LENGTH = ($LENGTH << 8) + ord(fread($this->socket, 1));
                            $LENGTH = ($LENGTH << 8) + ord(fread($this->socket, 1));
                        } else {
                            $LENGTH = ord(fread($this->socket, 1));
                            $LENGTH = ($LENGTH << 8) + ord(fread($this->socket, 1));
                            $LENGTH = ($LENGTH << 8) + ord(fread($this->socket, 1));
                            $LENGTH = ($LENGTH << 8) + ord(fread($this->socket, 1));
                        }
                    }
                }
            } else {
                $LENGTH = $BYTE;
            }
            // If we have got more characters to read, read them in.
            if ($LENGTH > 0) {
                $_      = "";
                $retlen = 0;
                while ($retlen < $LENGTH) {
                    $toread = $LENGTH - $retlen;
                    $_ .= fread($this->socket, $toread);
                    $retlen = strlen($_);
                }
                $RESPONSE[] = $_;
                $this->debug('>>> [' . $retlen . '/' . $LENGTH . '] bytes read.');
            }
            // If we get a !done, make a note of it.
```

```
            if ($_ == "!done")
                $receiveddone = true;
            $STATUS = socket_get_status($this->socket);
            if ($LENGTH > 0)
                $this->debug('>>> [' . $LENGTH . ', ' . $STATUS['unread_bytes'] . ']' . $_);
            if ((!$this->connected && !$STATUS['unread_bytes']) || ($this->connected && !$STATUS['unread_bytes'] && $receiveddone))
                break;
        }
        if ($parse)
            $RESPONSE = $this->parse_response($RESPONSE);
        return $RESPONSE;
    }



    /**
     * Write (send) data to Router OS
     *
     * @param string      $command    A string with the command to send
     * @param mixed       $param2     If we set an integer, the command will send this data as a "tag"
     *                                If we set it to boolean true, the funcion will send the comand and finish
     *                                If we set it to boolean false, the funcion will send the comand and wait for next command
     *                                Default: true
     *
     * @return boolean                Return false if no command especified
     */
    function write($command, $param2 = true)
    {
        if ($command) {
            $data = explode("\n", $command);
            foreach ($data as $com) {
                $com = trim($com);
                fwrite($this->socket, $this->encode_length(strlen($com)) . $com);
                $this->debug('<<< [' . strlen($com) . '] ' . $com);
            }
            if (gettype($param2) == 'integer') {
                fwrite($this->socket, $this->encode_length(strlen('.tag=' . $param2)) . '.tag=' . $param2 . chr(0));
                $this->debug('<<< [' . strlen('.tag=' . $param2) . '] .tag=' . $param2);
            } else if (gettype($param2) == 'boolean')
                fwrite($this->socket, ($param2 ? chr(0) : ''));
            return true;
        } else
            return false;
    }



    /**
     * Write (send) data to Router OS
```

```
     *
     * @param string      $com        A string with the command to send
     * @param array       $arr        An array with arguments or queries
     *
     * @return array                  Array with parsed
     */
    function comm($com, $arr = array())
    {
        $count = count($arr);
        $this->write($com, !$arr);
        $i = 0;
        foreach ($arr as $k => $v) {
            switch ($k[0]) {
                case "?":
                    $el = "$k=$v";
                    break;
                case "~":
                    $el = "$k~$v";
                    break;
                default:
                    $el = "=$k=$v";
                    break;
            }
            $last = ($i++ == $count - 1);
            $this->write($el, $last);
        }
        return $this->read();
    }
}
?>
```

Example 1

```
<?php

require('routeros_api.class.php');

$API = new routeros_api();

$API->debug = true;

if ($API->connect('111.111.111.111', 'LOGIN', 'PASSWORD')) {

   $API->write('/interface/getall');

   $READ = $API->read(false);
   $ARRAY = $API->parse_response($READ);
```

```
   print_r($ARRAY);

   $API->disconnect();

}

?>
```

OR

```
<?php

require('routeros_api.class.php');

$API = new routeros_api();

$API->debug = true;

if ($API->connect('111.111.111.111', 'LOGIN', 'PASSWORD')) {

   $API->write('/interface/getall');
   $ARRAY = $API->read();

   print_r($ARRAY);

   $API->disconnect();

}

?>
```

OR

```
<?php

require('routeros_api.class.php');

$API = new routeros_api();

$API->debug = true;

if ($API->connect('111.111.111.111', 'LOGIN', 'PASSWORD')) {

   $ARRAY = $API->comm('/interface/getall');
   print_r($ARRAY);

   $API->disconnect();

}
```

```
?>
```

Output

```
Array
(
   [0] => Array
      (
         [.id] => *1
         [name] => ether1
         [mtu] => 1500
         [type] => ether
         [running] => yes
         [dynamic] => no
         [slave] => no
         [comment] =>
         [disabled] => no
      )

   [1] => Array
      (
         [.id] => *2
         [name] => ether2
         [mtu] => 1500
         [type] => ether
         [running] => yes
         [dynamic] => no
         [slave] => no
         [comment] =>
         [disabled] => no
      )

   [2] => Array
      (
         [.id] => *3
         [name] => ether3
         [mtu] => 1500
         [type] => ether
         [running] => yes
         [dynamic] => no
         [slave] => no
         [comment] => ether3
         [disabled] => no
      )
)
```

Example 2

Thanks a lot for this API, It help me a lot to write my php page for our support team to have access to wireless registration table.

```
 $API->write('/interface/wireless/registration-table/print',false);
 $API->write('=stats=');
```

Output

```
Array
(
[0] =>
   Array
   (
   [.id] => *147
   [comment] =>
   [interface] => AP101
   [mac-address] => 00:0B:6B:37:58:33
   [ap] => false
   [wds] => false
   [rx-rate] => 11Mbps
   [tx-rate] => 11Mbps
   [packets] => 4043966,2669114
   [bytes] => 3961713942,280551024
   [frames] => 4043966,2669114
   [frame-bytes] => 3937477458,264536340
   [hw-frames] => 4500839,2669114
   [hw-frame-bytes] => 256326637,349947988
   [tx-frames-timed-out] => 0
   [uptime] => 1w13:09:12
   [last-activity] => 00:00:00.090
   [signal-strength] => -73dBm@11Mbps
   [signal-to-noise] => 30
   [strength-at-rates] => -73dBm@1Mbps 4m4s640ms,-73dBm@2Mbps 4m58s730ms,-73dBm@5.5Mbps 42s450ms,-73dBm@11Mbps 90ms
   [tx-ccq] => 91
   [p-throughput] => 5861
   [ack-timeout] => 31
   [last-ip] => 192.168.0.220
   [802.1x-port-enabled] => true
   [wmm-enabled] => false
   )
[1] => Array
   (
        ...
    )
...
)
```

Example 3

Adding vpn user

```
$API->comm("/ppp/secret/add", array(
          "name"     => "user",
          "password" => "pass",
          "remote-address" => "172.16.1.10",
          "comment"  => "{new VPN user}",
          "service"  => "pptp",
));
```

Example 4

Find registration-table id for specified MAC

```
$ARRAY = $API->comm("/interface/wireless/registration-table/print", array(
  ".proplist"=> ".id",
  "?mac-address" => "00:0E:BB:DD:FF:FF",
));
print_r($ARRAY);
```

Example 5

Count leases from specific IP Pool (using regexp count all IPs starting with 1.1.x.x)

```
$ARRAY = $API->comm("/ip/dhcp-server/lease/print", array(
    "count-only"=> "",
    "~active-address" => "1.1.",
));
print_r($ARRAY);
```

or

```
$API->write('/ip/dhcp-server/lease/print', false);
$API->write('=count-only=', false);
$API->write('~active-address~"1.1."');
$ARRAY = $API->read();
print_r($ARRAY);
```

Returns a number with leases

Article Sources and Contributors

API PHP class *Source*: http://wiki.mikrotik.com/index.php?oldid=22544 *Contributors*: Blaze, Bmenking, Chronos, Cristiandeluxe, Denis Basta, Janisk, JeremyWJ, Mangia, Mmorier, Mpapec, Newacct, Normis, Piotr.piwonski, Tiagoratto, Viktorc, Vitell