**cURL API calls with PHP, REST and JSON data (GET POST PUT DELETE)**

I was recently working on a project where I needed to integrate an external API using HTTP cURL requests. It was my first time doing this and I had a lot of problems figuring this out. I wrote this post so I can remember my cURL API calls for next time, and maybe it can help you as well.

The API calls and functions I’m using in this post are all working examples on PHP -v 5.6.

**PHP cURL Basics**

cURL stands for ‘Client URL Library’ and it allows you to connect and communicate with different types of servers with many different types of protocols (HTTP, https, FTP, proxy, cookies, …). More info about how cURL actually works can be found in [the official PHP documentation](https://www.php.net/manual/en/intro.curl.php). This article will provide more in-depth examples for integrating your applications.

I’ve received a lot of responses on ‘how does cURL actually work’ and I get the feeling that people don’t know what’s going on in a cURL call. Before we start with the article and our cURL setup, I’ve added a simple example of a plain cURL request. The request will return the API response as a string.

// create & initialize a curl session

$curl = curl\_init();

// set our url with curl\_setopt()

curl\_setopt($curl, CURLOPT\_URL, "api.example.com");

// return the transfer as a string, also with setopt()

curl\_setopt($curl, CURLOPT\_RETURNTRANSFER, 1);

// curl\_exec() executes the started curl session

// $output contains the output string

$output = curl\_exec($curl);

// close curl resource to free up system resources

// (deletes the variable made by curl\_init)

curl\_close($curl);

Note that we stored our curl\_exec() in a variable $output. This $output variable is still available in our program even after we closed it with curl\_close(). So after we did our call and closed the connection, we can still access the result using our $output variable.

Now that we understand the basics, let’s try to put this into a function we can reuse within our application.

**PHP cURL setup**

Implementing an external API into your project is probably going to take more than just one API call and from different pages in your project. This is why I’ve created a ‘simple’ PHP script that allows us to call this function, with a set of parameters, and a cURL request will be done.

Make sure to put this code into a file or place that can be accessed by your entire app or website. *(I’ve updated this function so we’ll be able to define the headers when we’re making the call. I’ve added a section for custom headers at the bottom!)*

**function** **callAPI**($method, $url, $data){

$curl = curl\_init();

**switch** ($method){

**case** "POST":

curl\_setopt($curl, CURLOPT\_POST, 1);

**if** ($data)

curl\_setopt($curl, CURLOPT\_POSTFIELDS, $data);

**break**;

**case** "PUT":

curl\_setopt($curl, CURLOPT\_CUSTOMREQUEST, "PUT");

**if** ($data)

curl\_setopt($curl, CURLOPT\_POSTFIELDS, $data);

**break**;

**default**:

**if** ($data)

$url = sprintf("%s?%s", $url, http\_build\_query($data));

}

// OPTIONS:

curl\_setopt($curl, CURLOPT\_URL, $url);

curl\_setopt($curl, CURLOPT\_HTTPHEADER, **array**(

'APIKEY: 111111111111111111111',

'Content-Type: application/json',

));

curl\_setopt($curl, CURLOPT\_RETURNTRANSFER, 1);

curl\_setopt($curl, CURLOPT\_HTTPAUTH, CURLAUTH\_BASIC);

// EXECUTE:

$result = curl\_exec($curl);

**if**(!$result){**die**("Connection Failure");}

curl\_close($curl);

**return** $result;

}

This is a basic setup for doing a cURL call and I’m using a switch statement to check if the API call will be a POST, PUT, or something else (get or delete). I’ll go deeper into the switch case while we’re doing the specific requests.

I’m using if-statements inside the switch-case to see if we want to provide JSON data into our call or not. For the POST and PUT request the if-statement is not really necessary because we’re only using POST or PUT with data, but it’s an extra security to make sure our call function won’t break.

**PHP cURL GET request**

The most simple API call is the GET call, so let’s start with that! Our callAPI function expects 3 parameters: $method, $url and $data. We need to give those parameters to all our API calls, so for a cURL GET, we can just set $data on false because we are not passing any data with a GET call.

$get\_data = callAPI('GET', 'https://api.example.com/get\_url/'.$user['User']['customer\_id'], **false**);

$response = json\_decode($get\_data, **true**);

$errors = $response['response']['errors'];

$data = $response['response']['data'][0];

$get\_data already returns all the data we want from the API in a json string. I’m using $response to convert the json string back to a usable PHP array. You can skip those steps if you want, this is my personal preference. I’m also using the extra $errors and $data arrays to store the actual data and errors.

**PHP cURL POST request**

Obviously, a POST request does require data. Make sure your json-data is correct, otherwise the request will keep returning errors. Although… If we receive errors from the API, that means our calls are working 😉

In my example I’m using the CakePHP syntax for setting up my json array, so don’t mind that.

$data\_array = **array**(

"customer" => $user['User']['customer\_id'],

"payment" => **array**(

"number" => **$this**->request->data['account'],

"routing" => **$this**->request->data['routing'],

"method" => **$this**->request->data['method']

),

);

$make\_call = callAPI('POST', 'https://api.example.com/post\_url/', json\_encode($data\_array));

$response = json\_decode($make\_call, **true**);

$errors = $response['response']['errors'];

$data = $response['response']['data'][0];

Because we’re doing an API call with json data, I’m converting my PHP array to a json string with json\_encode($data\_array);. The response will come in as a json string again, so I’m using json\_decode($make\_call, true); to convert the json string back to a usable PHP array. Same as we did in our GET call, so you can skip these steps again if you don’t need them.

**PHP cURL PUT request**

The PUT request is almost the same as the POST request. I had a hard time figuring out how to pass data into a PUT call. If we take a look at our callAPI() function, you see that I changed some things up between the PUT and the POST request. We can still use the same parameters in our callAPI() function as always.

$data\_array = **array**(

"amount" => (string)($lease['amount'] / $tenant\_count)

);

$update\_plan = callAPI('PUT', 'https://api.example.com/put\_url/'.$lease['plan\_id'], json\_encode($data\_array));

$response = json\_decode($update\_plan, **true**);

$errors = $response['response']['errors'];

$data = $response['response']['data'][0];

**PHP cURL DELETE request**

The delete request is very simple again. We can just hit the API url with the $id we want to remove and poof… it’s gone forever.

callAPI('DELETE', 'https://api.example.com/delete\_url/' . $id, **false**);

**What with flexible headers?**

In the beginning we defined our callAPI function with preset headers. But what if we, for some reason, need to change the headers a bit for another call? We don’t want to write a whole new callAPI function just to edit some headers. Therefore, here’s an option on how to make the preset headers flexible:

**function** **callAPI**($method, $url, $data, $headers = false){

$curl = curl\_init();

**switch** ($method){

...

}

// OPTIONS:

curl\_setopt($curl, CURLOPT\_URL, $url);

**if**(!$headers){

curl\_setopt($curl, CURLOPT\_HTTPHEADER, **array**(

'APIKEY: 111111111111111111111',

'Content-Type: application/json',

));

}**else**{

curl\_setopt($curl, CURLOPT\_HTTPHEADER, **array**(

'APIKEY: 111111111111111111111',

'Content-Type: application/json',

$headers

));

}

curl\_setopt($curl, CURLOPT\_RETURNTRANSFER, 1);

curl\_setopt($curl, CURLOPT\_HTTPAUTH, CURLAUTH\_BASIC);

// EXECUTE:

...

}

There are 2 differences here from our first function. 1: We’ve added an extra parameter in our function to define if we want to use a custom header or not. I put it to $headers = false to give it a default value. Now we’re not required to enter our headers with every call.

The second change is the if-statement when we’re setting the API headers. If we didn’t give in any headers when we make the call, it’s going to use our default headers instead of crashing. Now we’re ready to add custom headers with our call!

In this example, I’m using search parameters to search for specific data before I’ll pull in all the data with the API. To make the search, I obviously need to be able to add my search query into my callAPI headers. Here’s my example:

**Creating custom headers before our call**

$one\_month\_ago = date("Y-m-d", strtotime(date("Y-m-d", strtotime(date("Y-m-d"))) . "-1 month"));

$rent\_header = 'Search: and[][created][greater]=' . $one\_month\_ago . '%and[][created][less]=' . date('Y-m-d') . '%';

//the actual call with custom search header

$make\_call = callAPI('GET', 'https://api.example.com/get\_url/', **false**, $rent\_header);

This is just an example on how to add headers. My example is to get all rows where a rent was paid in the last 30 days. $one\_month\_ago is just a helper variable. The $rent\_header is the actual header I want to add to my default headers. This needs to be a string!!

When you’ve set the header, you can just do a regular api call and add your new header at the end.

I didn’t need to use any other API call methods like patch or purge or anything like that. These you need to figure out yourself. If there’s some magic going on in this post I hope my examples can give you a better understanding.

I recently wrote a part 2 for this post, that will talk about generating an AUTH-key (utoken) before we make our calls. [Make sure to check it out here as well!](https://weichie.com/blog/curl-api-calls-authentication/)