Getting off the Island (formerly Reaching out of Drupal)

TABLE OF CONTENTS

Overview

To communicate with external websites or web services we can make web requests via the Drupal::httpClient class. This is a wrapper for the Guzzle HTTP Client.

From https://www.php-fig.org/psr/psr-7/ HTTP messages are the foundation of web development. Web browsers and HTTP clients such as cURL create HTTP request messages that are sent to a web server, which provides an HTTP response message. Server-side code receives an HTTP request message, and returns an HTTP response message.

HTTP messages are typically abstracted from the end-user consumer, but as developers, we typically need to know how they are structured and how to access or manipulate them in order to perform our tasks, whether that might be making a request to an HTTP API, or handling an incoming request.

Guzzle utilizes PSR-7 as the HTTP message interface. PSR-7 describes common interfaces for representing HTTP messages. This allows Guzzle to work with any other library that utilizes PSR-7 message interfaces.

Guzzle example

This is an example which retrieves data from within a controller using a GET:

```
public function example1() {

  //Initialize client;
  $client = \Drupal::httpClient();
  $uri = 'https://demo.ckan.org/api/3/action/package_list';

  // Returns a GuzzleHttp\Psr7\Response.
  $response = $client->request('GET', 'https://demo.ckan.org/api/3/action/package_list');
  // Or using the magic method.
  $response = $client->get($uri);

  // Returns a GuzzleHttp\Psr7\Stream.
  $stream = $response->getBody();
  $json_data = Json::decode($stream);

  $help = $json_data['help'];
  $success = $json_data['success'];
  $result = $json_data['result'][0];

  $msg = "<br>URI: " . $uri;
  $msg .= "<br>Help: " . $help;
  $msg .= "<br>Success: " . $success;
  $msg .= "<br>Result: " . $result;

  $build['content'] = [
    '#type' => 'item',
    '#markup' => $this->t($msg),
  ];

  return $build;
}
```

Guzzle POST example

```
//Initialize client;
$client = \Drupal::httpClient();
$uri = 'http://demo.ckan.org/api/3/action/group_list';
$request = $client->post($uri, [
  'json' => [
    'id'=> 'data-explorer'
  ]
]);
$stream = $request->getBody();
$json_data = Json::decode($stream);
$help = $json_data['help'];
$success = $json_data['success'];
$result = $json_data['result'][0] . ' and ' . $json_data['result'][1];
```

Guzzle takes care of adding a 'Content-Type','application/json' header, as well as json_encoding the 'json' array.

Magic methods to send synchronous requests

```
$response = $client->get('http://httpbin.org/get');
$response = $client->delete('http://httpbin.org/delete');
$response = $client->head('http://httpbin.org/get');
$response = $client->options('http://httpbin.org/get');
$response = $client->patch('http://httpbin.org/patch');
$response = $client->post('http://httpbin.org/post');
$response = $client->put('http://httpbin.org/put');
```

From https://docs.guzzlephp.org/en/latest/quickstart.html#sending-requests

HTTP basic authentication

This shows a failed attempt to authenticate with Github's API with exception handling. It will log the error to Drupal's watchdog and display it on screen:

```
public function example2() {

  $msg = "";
  $client = \Drupal::httpClient();
  $uri = 'https://api.github.com/user';

  try {
    $request = $client->get($uri, [
      'auth' => ['username', 'password']
    ]);
    $response = $request->getBody();
    $msg .= "<br><strong>GET</strong>";
    $msg .= "<br>URI: " . $uri;
  }

  catch (ClientException $e) {
    \Drupal::messenger()->addError($e->getMessage());
    watchdog_exception('guzzle_examples', $e);
  }

  catch (\Exception $e) {
    \Drupal::messenger()->addError($e->getMessage());
    watchdog_exception('guzzle_examples', $e);
  }

  $build['content'] = [
    '#type' => 'item',
    '#markup' => $this->t($msg),
  ];

  return $build;
}
```

See https://docs.guzzlephp.org/en/latest/request-options.html#auth for more.

Exception handling

When using `Drupal::httpClient`, you should always wrap your requests in a try/catch block, to handle any exceptions. Here is an example of logging `Drupal::httpClient` request exceptions via `watchdog_exception`. This example will fail with a `401` error and display it on screen.

```
public function example2() {

  $msg = "";
  $client = \Drupal::httpClient();
  $uri = 'https://api.github.com/user';

  try {
    $request = $client->get($uri, [
      'auth' => ['username', 'password']
    ]);
    $response = $request->getBody();
    $msg .= "<br><strong>GET</strong>";
    $msg .= "<br>URI: " . $uri;
  }

  catch (ClientException $e) {
    \Drupal::messenger()->addError($e->getMessage());
    watchdog_exception('guzzle_examples', $e);
  }

  catch (\Exception $e) {
    \Drupal::messenger()->addError($e->getMessage());
    watchdog_exception('guzzle_examples', $e);
  }

  $build['content'] = [
    '#type' => 'item',
    '#markup' => $this->t($msg),
  ];

  return $build;
}
```

Guzzle Exceptions

You can get a full list of Exception types simply by listing the contents of the directory: `\<drupal_root\>/vendor/guzzlehttp/guzzle/src/Exception`. Utilizing this list allows you to provide different behavior based on exception type.

At the time of writing, the contents of that directory is:

- BadResponseException.php
- ClientException.php - use this to handle a 4xx error
- ConnectException.php
- GuzzleException.php
- RequestException.php
- SeekException.php
- ServerException.php
- TooManyRedirectsException.php
- TransferException.php

HTTP response status codes

From https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. Informational responses (100 – 199)
2. Successful responses (200 – 299)

3 Redirection messages (300 – 399)

4 Client error responses (400 – 499)

5 Server error responses (500 – 599)

Reading from an API

In this example, a class was created which extends SqlBase (docroot/core/modules/migrate/src/Plugin/migrate/source/SqlBase php). The code below is from the prepareRow() function which retrieves a row of data from the data source. In this case, rather than a SQL database, it is retrieved from an API. It uses basic http authentication during the GET call and if there are any errors, it updates a status elsewhere with a call to setUpdateStatus(). The following code is not included below, but it may be interesting to know what it does. If the GET succeeds, the data is parsed out and put into variables to be returned to the called. This acts just like prepareRow() does when retrieving a row from a SQL source. Taxonomy terms are looked up and added if they don't already exist (so taxonomy term id's can be returned) and the status is updated showing this row was successfully retrieved.

```
  $nard_auth_settings = Settings::get('nard_api_auth', []);
  $uri = $nard_auth_settings['default']['server'] . ':' . $nard_auth_settings['default']['port'];
  $uri .= '/sourcecontent/search';
  $client = \Drupal::httpClient();

  switch (strtolower($row->getSourceProperty('type'))) {
    case 'article':
      $uuid = $row->getSourceProperty('uuid');
      $id = $row->getSourceProperty('id');

      // Retrieve the article from the API.
      try {
        $response = $client->request('GET', $uri, [
          'auth' => [
            $nard_auth_settings['default']['username'],
            $nard_auth_settings['default']['password']
          ],
          'query' => [
            'uuid' => $row->getSourceProperty('uuid'),
            'properties' => 'Byline,updated,created,uuid',
          ],
          'timeout' => 3,
        ]);
      } catch (RequestException $e) {
        watchdog_exception('nard_myconnect', $e);
        \Drupal::logger('nard_myconnect')
          ->info("API error retrieving item $id with uuid=$uuid.");
        // TODO: Deal with this error
        $this->setUpdateStatus($uuid, 'error');
        return FALSE;
      }

      // Retrieve the details from the API call.
      try {
        $response = $response->getBody()->getContents();
      } catch (ClientException $e) {
        watchdog_exception('nard_myconnect', $e);
        \Drupal::logger('nard_myconnect')
          ->info("API error retrieving body for item $id with uuid=$uuid.");
        $this->setUpdateStatus($uuid, 'error');
        return FALSE;
      }

      // Parse retrieved JSON into fields.
      $response = json_decode($response, TRUE);
      if (!is_array($response)) {
        $response = [];
      }

  }
```

Curl Request using Drupal httpClient

From the now defunct link: http://btobac.com/blog/how-do-curl-request-using-drupal-httpclient-drupal-8

Here the author has an example of a function which takes a few parameters and can execute a POST, PUT or GET. There is no security code which you almost always need, but there is exception handling and error logging to Drupal watchdog.

Drupal HTTP client for curl HTTP request like POST, PUT, GET Method even for DELETE, you can add one type in the below switch case in the class method

```
class DrupalHTTPClient {

  public function initRequest($url, $headers = [],$content = "", $method = "POST", $msg = true, $type = "Third Party", $requestContentType = "json") {
    try {

      $client = \Drupal::httpClient();
      $params =  ['http_errors' => FALSE];
      $params[$requestContentType] = $content;

      $params = array_merge($headers,$params);

      switch ($method) {
        case 'POST':
          $response = $client->post($url, $params);
          break;
        case 'PUT':
          $response = $client->put($url, $params);
          break;
        case 'GET':
          $response = $client->get($url);
          break;
        default:
          $response = $client->post($url, $params);
          break;
      }
      if ($response->getStatusCode() == '200') {
        $data = $response->getBody()->getContents();
        //\Drupal::messenger()->addMessage(serialize($data));
        $result = json_decode($data, true);
        if($msg) {
          $setMessage = is_array($result) && isset($result['message']) ? $result['message'] :
            (is_string($data) ? $data : "Request is done successfully ");
            \Drupal::messenger()->addMessage($type.":".$setMessage);
        }
        if(isset($result['exception'])){
          $result_message = '';
          foreach($result as $key => $value){
            $result_message .= $key .': '.$value.', ';
          }
          \Drupal::logger('invalid_exception')->error(" \"Invalid Response: <i>`<strong>".$method."</strong> ".$url."`</i> resulted in a <strong>`status:".$response->getStatusCode().
          return NULL;
        }
        return $result;
      } else {
        \Drupal::messenger()->addError($type.' exception contact administrator');
        \Drupal::logger('server_exception')->error(" \"Server error: <i>`<strong>".$method."</strong> ".$url."`</i> resulted in a <strong>`".$response->getStatusCode()." ".$response
      }
    }
    catch (\GuzzleHttp\Exception\ConnectException $e) {
      \Drupal::messenger()->addError('Cannot contact '.$type);
      \Drupal::logger('connect_exception')->error(" \"Connection error: <i>`<strong>".$method."</strong> ".$url."`</i> resulted in a <strong>`".$e->getMessage()."`</strong> respon
    }
    return NULL;
  }

}

$cl = new DrupalHTTPClient();
```

```
$url = "http://api.adadasdadadasd/adasd/ad/ad";
$content = ["key" => "value"];
$data = $cl->initRequest($url,[],$content, "POST", TRUE, "anytime just flag");
```

Download a file using guzzle

Details are borrowed from https://gist.github.com/edutrul/9d04d7742545dbedd1a36f7b17632b7a

```
public function example3() {

  $msg = "";
  $client = \Drupal::httpClient();
  $uri = 'https://api.github.com/user';

  try {

    $source_uri = 'https://www.austinprogressivecalendar.com/sites/default/files/styles/medium/public/inserted-images/2018-04-02_5.jpg';
    // Note sites/default/files/abc directory must exist for this to succeed.
    $destination_uri = 'sites/default/files/abc/test.png';
    /** @var \GuzzleHttp\Psr7\Response $response */
    $response = $client->get($source_uri, ['sink' => $destination_uri]);
    // file gets downloaded to /sites/default/files/abc/test.png

    $msg .= "<br><strong>Retrieve File via Guzzle</strong>";
    $msg .= "<br>Source: " . $source_uri;
    $msg .= "<br>Dest: " . $destination_uri;
  }
  catch (ClientException $e) {
    \Drupal::messenger()->addError($e->getMessage());
    watchdog_exception('guzzle_examples', $e);
  }

  catch (\Exception $e) {
    \Drupal::messenger()->addError($e->getMessage());
    watchdog_exception('guzzle_examples', $e);
  }

  $build['content'] = [
    '#type' => 'item',
    '#markup' => $this->t($msg),
  ];

  return $build;
}
```

Download a file using curl in PHP

For comparison, I've included an example of how to download a file using curl.

From Stackoverflow: How to download a file using curl in php?

```
<?php

// Username or E-mail
$login = 'username';
// Password
$password = 'password';
// API Request
```

```php
$url = 'https://example.com/api';
// POST data
$data = array('someTask', 24);
// Convert POST data to json
$data_string = json_encode($data);
// initialize cURL
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL,$url);
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
curl_setopt($ch, CURLOPT_USERPWD, "$login:$password");
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "POST");
curl_setopt($ch, CURLOPT_POSTFIELDS, $data_string);
curl_setopt($ch, CURLOPT_HEADER, 1);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

// Execute cURL and store the response in a variable
$file = curl_exec($ch);

// Get the Header Size
$header_size = curl_getinfo($ch, CURLINFO_HEADER_SIZE);
// Get the Header from response
$header = substr($file, 0, $header_size);
// Get the Body from response
$body = substr($file, $header_size);
// Explode Header rows into an array
$header_items = explode("\n", $header);
// Close cURL handler
curl_close($ch);

// define new variable for the File name
$file_name = null;

// find the filname in the headers.
if(!preg_match('/filename="(.*?)"/', $header, $matches)){
  // If filename not found do something...
  echo "Unable to find filename.<br>Please check the Response Headers or Header parsing!";
  exit();
} else {
  // If filename was found assign the name to the variable above
  $file_name = $matches[1];
}
// Check header response, if HTTP response is not 200, then display the error.
if(!preg_match('/200/', $header_items[0])){
  echo '<pre>'.print_r($header_items[0], true).'</pre>';
  exit();
} else {
  // Check header response, if HTTP response is 200, then proceed further.

  // Set the header for PHP to tell it, we would like to download a file
  header('Content-Description: File Transfer');
  header('Content-Type: application/octet-stream');
  header('Content-Transfer-Encoding: binary');
  header('Expires: 0');
  header('Cache-Control: must-revalidate');
  header('Pragma: public');
  header('Content-Disposition: attachment; filename='.$file_name);

  // Echo out the file, which then should trigger the download
  echo $file;
  exit;
```

```
}
```

Also there were other examples on that page that are probably worth looking at.

Someone responded with this shorter paste in November 2020 at http://paste.debian.net/1170460/ with the following comment: this code is not downloading a file with PHP, it is PROXYING a file with PHP, and it's not doing a good job either, being way slower and more memory-hungry than required, this script would do it much faster, benchmark it

```php
<?php

declare(strict_types=1);
$ch = curl_init();
curl_setopt_arrah($ch, [
  CURLOPT_URL => 'http://example.org',
  CURLOPT_HEADERFUNCTION => function ($ch, string $header): int {
    $header_trimmed = trim($header);
    if (strlen($header_trimmed) > 0) {
      header($header, FALSE);
    }
    return strlen($header);
  }
]);
header('Content-Description: File Transfer');
header('Content-Type: application/octet-stream');

curl_exec($ch);
curl_close($ch);
```

Resources

- Guzzle docs https://docs.guzzlephp.org/en/stable/overview.html
- Guzzle project https://github.com/guzzle/guzzle
- Article at Drupalize.me by William Hetherington from December 2015 on using Drupal 8 to speak http (using guzzle): https://drupalize.me/blog/201512/speak-http-drupal-httpclient
- Nice little code snippet from J M Olivas showing how to use dependency injection with Guzzle at https://gist.github.com/jmolivas/ca258d7f2742d9e1aae4
- PSR-7: HTTP message interfaces describes common interfaces for representing HTTP messages and URIs https://www.php-fig.org/psr/psr-7/

Back to top

Page last modified: Apr 14 2023.

Edit this page on GitHub

This site uses Just the Docs, a documentation theme for Jekyll.