Guzzle 6 (index.html)

TABLE OF CONTENTS

Search...

Quickstart

This page provides a quick introduction to Guzzle and introductory examples. If you have not already installed, Guzzle, head over to the Installation (overview.html#installation) page.

Making a Request

You can send requests with Guzzle using a `GuzzleHttp\ClientInterface` object.

Creating a Client

```
use GuzzleHttp\Client;

$client = new Client([
    // Base URI is used with relative requests
    'base_uri' => 'http://httpbin.org',
    // You can set any number of default request options.
    'timeout'  => 2.0,
]);
```

Clients are immutable in Guzzle 6, which means that you cannot change the defaults used by a client after it's created.

The client constructor accepts an associative array of options:

`base_uri`

(string|UriInterface) Base URI of the client that is merged into relative URIs. Can be a string or instance of UriInterface. When a relative URI is provided to a client, the client will combine the base URI with the relative URI using the rules described in RFC 3986, section 2 (http://tools.ietf.org/html/rfc3986#section-5.2).

```
// Create a client with a base URI
$client = new GuzzleHttp\Client(['base_uri' => 'https://
// Send a request to https://foo.com/api/test
$response = $client->request('GET', 'test');
// Send a request to https://foo.com/root
$response = $client->request('GET', '/root');
```

Don't feel like reading RFC 3986? Here are some quick examples on how a `base_uri` is resolved with another URI.

base_uri	URI	Result
`http://foo.com`	`/bar`	`http://foo.com/bar`

v: stable

base_uri	URI	Result
`http://foo.com/foo`	`/bar`	`http://foo.com/bar`
`http://foo.com/foo`	`bar`	`http://foo.com/bar`
`http://foo.com/foo/`	`bar`	`http://foo.com/foo/bar`
`http://foo.com`	`http://baz.com`	`http://baz.com`
`http://foo.com/?bar`	`bar`	`http://foo.com/bar`

`handler`

(callable) Function that transfers HTTP requests over the wire. The function is called with a `Psr7\Http\Message\RequestInterface` and array of transfer options, and must return a `GuzzleHttp\Promise\PromiseInterface` that is fulfilled with a `Psr7\Http\Message\ResponseInterface` on success. `handler` is a constructor only option that cannot be overridden in per/request options.

`...`

(mixed) All other options passed to the constructor are used as default request options with every request created by the client.

Sending Requests

Magic methods on the client make it easy 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');
```

You can create a request and then send the request with the client when you're ready:

```
use GuzzleHttp\Psr7\Request;

$request = new Request('PUT', 'http://httpbin.org/put');
$response = $client->send($request, ['timeout' => 2]);
```

Client objects provide a great deal of flexibility in how request are transferred including default request options, default handler stack middleware that are used by each request, and a base URI that allows you to send requests with relative URIs.

You can find out more about client middleware in the Handlers and Middleware (handlers-and-middleware.html) page of the documentation.

Async Requests

You can send asynchronous requests using the magic methods provided by a client:

```
$promise = $client->getAsync('http://httpbin.org/get');
$promise = $client->deleteAsync('http://httpbin.org/delete')
$promise = $client->headAsync('http://httpbin.org/get');
$promise = $client->optionsAsync('http://httpbin.org/get');
$promise = $client->patchAsync('http://httpbin.org/patch');
$promise = $client->postAsync('http://httpbin.org/post');
$promise = $client->putAsync('http://httpbin.org/put');
```

You can also use the sendAsync() and requestAsync() methods of a client:

```
use GuzzleHttp\Psr7\Request;

// Create a PSR-7 request object to send
$headers = ['X-Foo' => 'Bar'];
$body = 'Hello!';
$request = new Request('HEAD', 'http://httpbin.org/head', $h

// Or, if you don't need to pass in a request instance:
$promise = $client->requestAsync('GET', 'http://httpbin.org/
```

The promise returned by these methods implements the Promises/A+ spec (https://promisesaplus.com/), provided by the Guzzle promises library (https://github.com/guzzle/promises). This means that you can chain `then()` calls off of the promise. These then calls are either fulfilled with a successful `Psr\Http\Message\ResponseInterface` or rejected with an exception.

```
use Psr\Http\Message\ResponseInterface;
use GuzzleHttp\Exception\RequestException;

$promise = $client->requestAsync('GET', 'http://httpbin.org/
$promise->then(
    function (ResponseInterface $res) {
        echo $res->getStatusCode() . "\n";
    },
    function (RequestException $e) {
        echo $e->getMessage() . "\n";
        echo $e->getRequest()->getMethod();
    }
);
```

Concurrent requests

v: stable

You can send multiple requests concurrently using promises and asynchronous requests.

```
use GuzzleHttp\Client;
use GuzzleHttp\Promise;

$client = new Client(['base_uri' => 'http://httpbin.org/']);

// Initiate each request but do not block
$promises = [
    'image' => $client->getAsync('/image'),
    'png'   => $client->getAsync('/image/png'),
    'jpeg'  => $client->getAsync('/image/jpeg'),
    'webp'  => $client->getAsync('/image/webp')
];

// Wait on all of the requests to complete. Throws a Connect
// if any of the requests fail
$results = Promise\unwrap($promises);

// Wait for the requests to complete, even if some of them f
$results = Promise\settle($promises)->wait();

// You can access each result using the key provided to the
// function.
echo $results['image']['value']->getHeader('Content-Length')
echo $results['png']['value']->getHeader('Content-Length')[0
```

You can use the `GuzzleHttp\Pool` object when you have an indeterminate amount of requests you wish to send.

```php
use GuzzleHttp\Pool;
use GuzzleHttp\Client;
use GuzzleHttp\Psr7\Request;

$client = new Client();

$requests = function ($total) {
    $uri = 'http://127.0.0.1:8126/guzzle-server/perf';
    for ($i = 0; $i < $total; $i++) {
        yield new Request('GET', $uri);
    }
};

$pool = new Pool($client, $requests(100), [
    'concurrency' => 5,
    'fulfilled' => function ($response, $index) {
        // this is delivered each successful response
    },
    'rejected' => function ($reason, $index) {
        // this is delivered each failed request
    },
]);

// Initiate the transfers and create a promise
$promise = $pool->promise();

// Force the pool of requests to complete.
$promise->wait();
```

Or using a closure that will return a promise once the pool calls the closure.

```php
$client = new Client();

$requests = function ($total) use ($client) {
    $uri = 'http://127.0.0.1:8126/guzzle-server/perf';
    for ($i = 0; $i < $total; $i++) {
        yield function() use ($client, $uri) {
            return $client->getAsync($uri);
        };
    }
};

$pool = new Pool($client, $requests(100));
```

Using Responses

In the previous examples, we retrieved a `$response` variable or we were delivered a response from a promise. The response object implements a PSR-7 response, `Psr\Http\Message\ResponseInterface`, and contains lots of helpful information.

You can get the status code and reason phrase of the response:

```
$code = $response->getStatusCode(); // 200
$reason = $response->getReasonPhrase(); // OK
```

You can retrieve headers from the response:

```
// Check if a header exists.
if ($response->hasHeader('Content-Length')) {
    echo "It exists";
}

// Get a header from the response.
echo $response->getHeader('Content-Length');

// Get all of the response headers.
foreach ($response->getHeaders() as $name => $values) {
    echo $name . ': ' . implode(', ', $values) . "\r\n";
}
```

The body of a response can be retrieved using the `getBody` method. The body can be used as a string, cast to a string, or used as a stream like object.

```
$body = $response->getBody();
// Implicitly cast the body to a string and echo it
echo $body;
// Explicitly cast the body to a string
$stringBody = (string) $body;
// Read 10 bytes from the body
$tenBytes = $body->read(10);
// Read the remaining contents of the body as a string
$remainingBytes = $body->getContents();
```

Query String Parameters

You can provide query string parameters with a request in several ways.

You can set query string parameters in the request's URI:

```
$response = $client->request('GET', 'http://httpbin.org?foo=
```

You can specify the query string parameters using the `query` request option as an array.

```
$client->request('GET', 'http://httpbin.org', [
    'query' => ['foo' => 'bar']
]);
```

Providing the option as an array will use PHP's `http_build_query` function to format the query string.

And finally, you can provide the `query` request option as a string.

```
$client->request('GET', 'http://httpbin.org', ['query' => 'f
```

Uploading Data

Guzzle provides several methods for uploading data.

You can send requests that contain a stream of data by passing a string, resource returned from `fopen` , or an instance of a `Psr\Http\Message\StreamInterface` to the `body` request option.

```
// Provide the body as a string.
$r = $client->request('POST', 'http://httpbin.org/post', [
    'body' => 'raw data'
]);

// Provide an fopen resource.
$body = fopen('/path/to/file', 'r');
$r = $client->request('POST', 'http://httpbin.org/post', ['b

// Use the stream_for() function to create a PSR-7 stream.
$body = \GuzzleHttp\Psr7\stream_for('hello!');
$r = $client->request('POST', 'http://httpbin.org/post', ['b
```

An easy way to upload JSON data and set the appropriate header is using the `json` request option:

```
$r = $client->request('PUT', 'http://httpbin.org/put', [
    'json' => ['foo' => 'bar']
]);
```

POST/Form Requests

In addition to specifying the raw data of a request using the `body` request option, Guzzle provides helpful abstractions over sending POST data.

Sending form fields

Sending `application/x-www-form-urlencoded` POST requests requires that you specify the POST fields as an array in the `form_params` request options.

```
$response = $client->request('POST', 'http://httpbin.org/pos
    'form_params' => [
        'field_name' => 'abc',
        'other_field' => '123',
        'nested_field' => [
            'nested' => 'hello'
        ]
    ]
]);
```

Sending form files

You can send files along with a form (`multipart/form-data` POST requests), using the `multipart` request option. `multipart` accepts an array of associative arrays, where each associative array contains the following keys:

- name: (required, string) key mapping to the form field name.
- contents: (required, mixed) Provide a string to send the contents of the file as a string, provide an fopen resource to stream the contents from a PHP stream, or provide a `Psr\Http\Message\StreamInterface` to stream the contents from a PSR-7 stream.

```
$response = $client->request('POST', 'http://httpbin.org/pos
    'multipart' => [
        [
            'name'     => 'field_name',
            'contents' => 'abc'
        ],
        [
            'name'     => 'file_name',
            'contents' => fopen('/path/to/file', 'r')
        ],
        [
            'name'     => 'other_file',
            'contents' => 'hello',
            'filename' => 'filename.txt',
            'headers'  => [
                'X-Foo' => 'this is an extra header to inclu
            ]
        ]
    ]
]);
```

Cookies

Guzzle can maintain a cookie session for you if instructed using the `cookies` request option. When sending a request, the `cookies` option must be set to an instance of `GuzzleHttp\Cookie\CookieJarInterface` .

```
// Use a specific cookie jar
$jar = new \GuzzleHttp\Cookie\CookieJar;
$r = $client->request('GET', 'http://httpbin.org/cookies', [
    'cookies' => $jar
]);
```

You can set `cookies` to `true` in a client constructor if you would like to use a shared cookie jar for all requests.

```
// Use a shared client cookie jar
$client = new \GuzzleHttp\Client(['cookies' => true]);
$r = $client->request('GET', 'http://httpbin.org/cookies');
```

Redirects

Guzzle will automatically follow redirects unless you tell it not to. You can customize the redirect behavior using the `allow_redirects` request option.

- Set to `true` to enable normal redirects with a maximum number of 5 redirects. This is the default setting.
- Set to `false` to disable redirects.
- Pass an associative array containing the 'max' key to specify the maximum number of redirects and optionally provide a 'strict' key value to specify whether or not to use strict RFC compliant redirects (meaning redirect POST requests with POST requests vs. doing what most browsers do which is redirect POST requests with GET requests).

```
$response = $client->request('GET', 'http://github.com');
echo $response->getStatusCode();
// 200
```

The following example shows that redirects can be disabled.

```
$response = $client->request('GET', 'http://github.com', [
    'allow_redirects' => false
]);
echo $response->getStatusCode();
// 301
```

Exceptions

Guzzle throws exceptions for errors that occur during a transfer.

- In the event of a networking error (connection timeout, DNS errors, etc.), a `GuzzleHttp\Exception\RequestException` is thrown. This exception extends from

`GuzzleHttp\Exception\TransferException` . Catching this exception will catch any exception that can be thrown while transferring requests.

```
use GuzzleHttp\Psr7;
use GuzzleHttp\Exception\RequestException;

try {
    $client->request('GET', 'https://github.com/_abc_12
} catch (RequestException $e) {
    echo Psr7\str($e->getRequest());
    if ($e->hasResponse()) {
        echo Psr7\str($e->getResponse());
    }
}
```

- A `GuzzleHttp\Exception\ConnectException` exception is thrown in the event of a networking error. This exception extends from `GuzzleHttp\Exception\RequestException` .
- A `GuzzleHttp\Exception\ClientException` is thrown for 400 level errors if the `http_errors` request option is set to true. This exception extends from `GuzzleHttp\Exception\BadResponseException` and `GuzzleHttp\Exception\BadResponseException` extends from `GuzzleHttp\Exception\RequestException` .

```
use GuzzleHttp\Exception\ClientException;

try {
    $client->request('GET', 'https://github.com/_abc_12
} catch (ClientException $e) {
    echo Psr7\str($e->getRequest());
    echo Psr7\str($e->getResponse());
}
```

- A `GuzzleHttp\Exception\ServerException` is thrown for 500 level errors if the `http_errors` request option is set to true. This exception extends from `GuzzleHttp\Exception\BadResponseException` .
- A `GuzzleHttp\Exception\TooManyRedirectsException` is thrown when too many redirects are followed. This exception extends from `GuzzleHttp\Exception\RequestException` .

All of the above exceptions extend from `GuzzleHttp\Exception\TransferException` .

Environment Variables

Guzzle exposes a few environment variables that can be used to customize the behavior of the library.

`GUZZLE_CURL_SELECT_TIMEOUT`

Controls the duration in seconds that a curl_multi_* handler will use when selecting on curl handles using `curl_multi_select()`. Some systems have issues with PHP's implementation of `curl_multi_select()` where calling this function always results in waiting for the maximum duration of the timeout.

`HTTP_PROXY`

Defines the proxy to use when sending requests using the "http" protocol.

Note: because the HTTP_PROXY variable may contain arbitrary user input on some (CGI) environments, the variable is only used on the CLI SAPI. See https://httpoxy.org (https://httpoxy.org) for more information.

`HTTPS_PROXY`

Defines the proxy to use when sending requests using the "https" protocol.

Relevant ini Settings

Guzzle can utilize PHP ini settings when configuring clients.

`openssl.cafile`

Specifies the path on disk to a CA file in PEM format to use when sending requests over "https". See: https://wiki.php.net/rfc/tls-peer-verification#phpini_defaults (https://wiki.php.net/rfc/tls-peer-verification#phpini_defaults)

Overview (overview.html)

Request Options (request-options.html)