By https://andreiabohner.org

Consume APIs in a snap!

HttpClient

Provides utilities to consume APIs

Symfony 4.3

Install

HttpClient is a standalone package

```
$ composer require symfony/http-client
```

Create the low-level HTTP client that makes requests

Using the HttpClient

```
use Symfony\Component\HttpClient\HttpClient;
```

Create Options: options defined here are added to all requests made by this client

default values

```
$httpClient = HttpClient::create([], 6, 50);
```

max host connections (optional)

max pending pushes (optional)(only cURL)

Request Options: here you can define options that apply only to this request (overrides any global option defined by the HTTPclient::create)

The request() method perform all kinds of HTTP requests

```
$response = $httpClient->request('GET', 'https://symfony.com/versions.json', []);
```

code execution continues immediately, it doesn't wait to receive the response

HTTP method

URL

```
$statusCode = $response->getStatusCode();
```

returns the status code
E.g.: 200

getting the response headers
waits until they arrive

```
$contentType = $response->getHeaders()['content-type'][0];
```

returns: 'application/json'

```
$content = $response->getContent();
```

getting the response contents will block the execution until the full response contents are received (use streaming responses for full async apps)

returns:
{"lts":"3.4.28","latest":"4.2.9","dev":"4.3.0-RC1",...}

```
$content = $response->toArray();
```

returns:
["lts" => "3.4.28", "latest" => "4.2.9",
"dev" => "4.3.0-RC1", "2.0"=>"2.0.25", ...]

Only supported when using cURL

HTTP/2 request

HTTP/2 will be used by default if:
* cURL-based transport used
* libcurl version is >= 7.36.0
* request using HTTPs protocol

To enable for HTTP requests:

```
$httpClient = HttpClient::create(['http_version' => '2.0']);
```

HTTP/2 PUSH support

Available when:
* libcurl >= 7.61.0 is used
* PHP >= 7.2.17 / 7.3.4

Pushed responses are put into a temporary cache and are used when a subsequent request is triggered for the corresponding URLs.

HTTPClient supports native PHP streams and cURL

HttpClient::create() selects cURL transport if cURL PHP extension is enabled and falls back to PHP streams otherwise.

Explicitly selecting the transport

```
use Symfony\Component\HttpClient\CurlHttpClient;
use Symfony\Component\HttpClient\NativeHttpClient;

// native PHP streams
$httpClient = new NativeHttpClient();

// cURL PHP extension
$httpClient = new CurlHttpClient();
```

HttpClient

Symfony 4.3

Options for Create and Request

	option	default value	definition and examples
authentication	`auth_basic`	`null`	An array containing the username as first value, and optionally the password as the second one; or string like username:password - enabling HTTP Basic authentication (RFC 7617).
	`auth_bearer`	`null`	A token enabling HTTP Bearer authorization (RFC 6750).

Use the same authentication for all requests

```
$httpClient = HttpClient::create([
    'auth_basic' => ['the-username'],
    'auth_basic' => ['the-username', 'the-password'],
    'auth_bearer' => 'the-bearer-token',
]);
```

HTTP Basic authentication with only the username

HTTP Basic authentication with username and password

HTTP Bearer authentication (also called token authentication)

```
$response = $httpClient->request('GET', 'https://...', [
    'auth_basic' => ['the-username', 'the-password'],
]);
```

use a different HTTP Basic authentication only for this request

query string params	`query`	`[]`	Associative array of query string values to merge with the request's URL.

```
$response = $httpClient->request('GET', 'https://httpbin.org/get', [
    'query' => [
        'token' => '...',
        'name' => '...',
    ],
]);
```

these values are automatically encoded before including them in the URL

setting HTTP headers	`headers`	`[]`	Headers names provided as keys or as part of values.

```
$httpClient = HttpClient::create(['headers' => [
    'User-Agent' => 'My Fancy App',
]]);
```

header added to all requests made by this client

```
$response = $httpClient->request('POST', 'https://...', [
    'headers' => [
        'Content-Type' => 'text/plain',
    ],
]);
```

header only included in this request and overrides the value of the same header if defined globally by create()

uploading data	`body`	`''`	You can use regular strings, closures, iterables and resources to upload data. They'll be processed automatically when making the requests.

```
$response = $httpClient->request('POST', 'https://...', [
    'body' => 'raw data',
    'body' => ['parameter1' => 'value1', '...'],
    'body' => function () {
        // ...
    },
    'body' => fopen('/path/to/file', 'r'),
]);
```

using a regular string

using an array of parameters

using a closure to generate the uploaded data

using a resource to get the data from it

json payload	`json`	`null`	When uploading JSON payloads, use the json option instead of body. The given content will be JSON-encoded automatically and the request will add the Content-Type: application/json automatically too.

```
$response = $httpClient->request('POST', 'https://...', [
    'json' => ['param1' => 'value1', '...'],
]);
```

`user_data`	`null`	Any extra data to attach to the request (scalar, callable, object...) that must be available via $response->getInfo('user_data') - not used internally.
`max_redirects`	`20`	The maximum number of redirects to follow; a value lower or equal to zero means redirects should not be followed; "Authorization" and "Cookie" headers must not follow except for the initial host name. If the number of redirects is higher than the configured value, you'll get a RedirectionException.
`http_version`	`null`	Defaults to the best supported version, typically 1.1 or 2.0.
`base_uri`	`null`	The URI to resolve relative URLs, following rules in RFC 3986, section 2 .
`buffer`	`true`	Whether the content of the response should be buffered or not.

HttpClient

PSR-18 compatible

Symfony 4.3

option	default value	definition and examples
on_progress	null	Details about the response progress (e.g. display a progress bar) / abort a request throwing any exceptions.

```
$url = 'https://releases.ubuntu.com/18.04.1/ubuntu-18.04.1-desktop-amd64.iso';
$response = $httpClient->request('GET', $url, [

    'buffer' => false,

    'on_progress' => function (int $dlNow, int $dlSize, array $info): void {
        // ...
    },
]);
```

optional: if you don't want to buffer the response in memory

optional: to display details about the response progress

resolve	[]	A map of host to IP address that should replace DNS resolution. Protect webhooks against calls to internal endpoints.
proxy	null	Get through an HTTP proxy. By default, the proxy-related env vars handled by cURL should be honored.
no_proxy	null	A comma separated list of hosts that do not require a proxy to be reached.
timeout	null	The inactivity timeout - defaults to ini_get('default_socket_timeout').
bindto	0	The interface or the local socket to bind to.
verify_peer	true	Require verification of SSL certificate used.
verify_host	true	
cafile	null	Location of Certificate Authority file on local filesystem which should be used with the verify_peer context option to authenticate the identity of the remote peer.
capath	null	If cafile is not specified or if the certificate is not found there, the directory pointed to by capath is searched for a suitable certificate. capath must be a correctly hashed certificate directory.
local_cert	null	Path to local certificate file on filesystem.
local_pk	null	Path to local private key file on filesystem in case of separate files for certificate (local_cert) and private key.
passphrase	null	Passphrase with which your local_cert file was encoded.
ciphers	null	Sets the list of available ciphers.
peer_fingerprint	null	Pin public keys of remote certificates. Aborts when the remote certificate digest doesn't match the specified hash.
capture_peer_cert_chain	false	If set to TRUE a peer_certificate_chain context option will be created containing the certificate chain.
extra	[]	Additional options that can be ignored if unsupported, unlike regular options

SSL / certificates (https://php.net/context.ssl)

Cookies

HTTPClient is stateless so it doesn't handle cookies automatically. You can:
- handle cookies yourself using the Cookie HTTP header
- use the BrowserKit component which provides this feature and integrates seamlessly with the HttpClient component

Caching Requests and Responses

The CachingHttpClient decorator allows caching responses and serving them from the local storage for next requests.
The implementation leverages the HttpCache class under the hood so that the HttpKernel component needs to be installed in your app.

```
use Symfony\Component\HttpClient\HttpClient;
use Symfony\Component\HttpClient\CachingHttpClient;
use Symfony\Component\HttpKernel\HttpCache\Store;

$store = new Store('/path/to/cache/storage/');
$client = HttpClient::create();
$client = new CachingHttpClient($client, $store);

$response = $client->request('GET', 'https://example.com/cacheable-resource');
```

accepts a third argument to set the options for HttpCache

won't hit the network if the resource is already in the cache

By https://andreiabohner.org

HttpClient

Supports synchronous and asynchronous operations

Symfony 4.3

The response is an object of type ResponseInterface → Response

Responses are always asynchronous: the call to the method returns immediately instead of waiting to receive the response

Response Methods

```
$response    = $httpClient->request('GET', 'https://...');

$statusCode = $response->getStatusCode();
```
returns the HTTP status code of the response

```
$headers     = $response->getHeaders();
```
gets the HTTP headers as string[][] with the header names lower-cased

```
$content     = $response->getContent();
```
gets the response body as a string

```
$httpInfo    = $response->getInfo();
```
gets info coming from the transport layer

```
$startTime   = $response->getInfo('start_time');
```
gets individual info

Info coming from the transport layer

$response->getInfo() Options

- user_data
- response_headers
- debug
- url
- error
- http_method
- http_code
- redirect_count
- start_time
- connect_time
- redirect_time
- starttransfer_time
- total_time
- namelookup_time
- size_upload
- size_download
- primary_ip
- primary_port
- redirect_url

gets detailed logs about the HTTP transaction

E.g.:
`$response->getInfo('debug')`

Streaming Responses

for full async apps

```
$url = 'https://releases.ubuntu.com/18.04.1/ubuntu-18.04.1-desktop-amd64.iso';
$response = $httpClient->request('GET', $url, [
    'buffer' => false,
    'on_progress' => function (int $dlNow, int $dlSize, array $info): void {
        // ...
    },
]);

if (200 !== $response->getStatusCode()) {
    throw new \Exception('...');
}

$fileHandler = fopen('/ubuntu.iso', 'w');

foreach ($httpClient->stream($response, 0.0) as $chunk) {
    fwrite($fileHandler, $chunk->getContent());
}
```

responses are lazy: this code is executed as soon as headers are received

(optional) max number of seconds to wait before yelding a timeout chunk

get the response contents in chunk

stream : get chunks of the response sequentially instead of waiting for the entire response

response chunks implement `Symfony\Contracts\HttpClient\ChunkInterface`

Scoping Client

autoconfigure the HTTP client based on the requested URL

HTTP client options that depend on the URL of the request

```
use Symfony\Component\HttpClient\HttpClient;
use Symfony\Component\HttpClient\ScopingHttpClient;

$client = HttpClient::create();
$httpClient = new ScopingHttpClient($client, [
    'https://api\.github\.com/' => [
        'headers' => [
            'Accept' => 'application/vnd.github.v3+json',
            'Authorization' => 'token '.$githubToken,
        ],
        'base_uri' => 'https://api.github.com/',
    ],
], 'https://api\.github\.com/'
);
```

the key is a regexp which must match the beginning of the request URL

the options defined as values apply only to the URLs matching the regular expressions defined as key

the (optional) 3rd argument is the regexp applied to all relative URLs (when using base_uri)

HttpClient

Symfony 4.3

you can configure multiple clients with different configurations and inject them into your services

Symfony Framework Integration

```yaml
# config/packages/framework.yaml
framework:
    # ...
    http_client:
        max_redirects: 7
        max_host_connections: 10
```

Use the `http_client` key to configure the default HTTP client used in the app

```yaml
# config/packages/framework.yaml
framework:
    # ...
    http_client:
        scoped_clients:
            crawler.client:
                headers: [{ 'X-Powered-By': 'ACME App' }]
                http_version: '1.0'
            some_api.client:
                max_redirects: 7
```

Defining multiple http_clients

Injecting the HTTP Client into Services

One HTTP client

```php
use Symfony\Contracts\HttpClient\HttpClientInterface;

class SomeService
{
    private $httpClient;

    public function __construct(HttpClientInterface $httpClient)
    {
        $this->httpClient = $httpClient;
    }
}
```

inject the HTTP client into any service by type-hinting a constructor argument with the `HttpClientInterface`

Multiple HTTP clients

```yaml
# config/services.yaml
services:
    # ...

    # whenever a service type-hints HttpClientInterface, inject the GitHub client
    Symfony\Contracts\HttpClient\HttpClientInterface: '@api_client.github'

    # inject the HTTP client called 'crawler' into this argument of this service
    App\Some\Service:
        $someArgument: '@http_client.crawler'
```

you can choose the service using any available method in Symfony

Handling Exceptions

When the HTTP status code of the response is in the 300-599 range (i.e. 3xx, 4xx or 5xx) your code is expected to handle it. If you don't do that, the getHeaders() and getContent() methods throw an appropriate exception:

```php
// the response of this request will be a 403 HTTP error
$response = $httpClient->request('GET', 'https://httpbin.org/status/403');

// this code results in a Symfony\Component\HttpClient\Exception\ClientException
// because it doesn't check the status code of the response
$content = $response->getContent();

// pass FALSE as the optional argument to not throw an exception and return
// instead the original response content (even if it's an error message)
$content = $response->getContent(false);
```