



# HttpClient

### Symfony 4.3

# Options for Create and Request

authentication	option auth_basic	default value null	definition and examples 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). \$ top for all requests for all requ	
			<pre>'auth_basic' =&gt; ['the-username', 'the-password'], ]); Basic authentication only for this request</pre>	
query string params	query	[]	<pre>Associative array of query string values to merge with the request's URL. \$response = \$httpClient-&gt;request('GET', 'https://httpbin.org/get', [             'query' =&gt; [             'token' =&gt; '',            these values are automatically             'name' =&gt; '',            these values are automatically             in ame ' =&gt; '',            these values are automatically             in ame ' =&gt; '',</pre>	
etting HTTP headers	headers	[]	<pre>Headers names provided as keys or as part of values. \$httpClient = HttpClient::create(['headers' =&gt; [     'User-Agent' =&gt; 'My Fancy App',</pre>	
setting			<pre>\$response = \$httpClient-&gt;request('POST', 'https://', [</pre>	
uploading data	body	0	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', using a regular string 'body' => ['parameter1' => 'value1', ''], using an array 'body' => function () { // using a closure to generate }, using a resource to generate the uploaded data 'body' => fopen('/path/to/file', 'r'), using a resource to get the data from it	
json payload	json	null	<pre>]); // get the data from it 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-&gt;request('POST', 'https://', [</pre>	
	user_data	null	Any extra data to attach to the request (scalar, callable, object) that must be available via <b>\$response-&gt;getInfo('user_data')</b> - not used internally.	
	<pre>max_redirects</pre>	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.	

By https://andreiabohner.org

# HttpClient PSR-18 compatible

# Symfony

	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.		
			<pre>\$url = 'https://releases.ubuntu.com/18.04.1/ubuntu-18.04.1-desktop-amd64.iso'; \$response = \$httpClient-&gt;request('GET', \$url, [</pre>		
			'buffer' => false, optional: if you don't want to buffer the response in memory		
			<pre>'on_progress' =&gt; function (int \$dlNow, int \$dlSize, array \$info): void {     //     poptional: to display details     about the response progress</pre>		
	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.		
(100.	verify_peer	true	Require verification of SSL certificate used.		
	verify_host	true			
וואיויריליר	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.		
100	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		
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); accepts a third argument to set the options for HttpCache

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

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



HttpClier

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

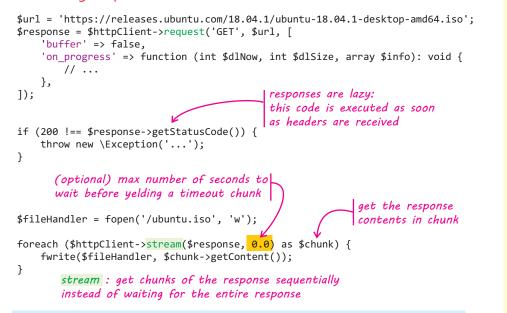
**Response Methods** 

The response is an object of type Response ResponseInterface

Info coming from the transport layer

\$response	<pre>= \$httpClient-&gt;request('GET', 'https</pre>	<pre>\$response-&gt;getInfo() Options</pre>	
\$statusCode	<pre>= \$response-&gt;getStatusCode();</pre>	returns the HTTP status code of the response	user_data is non-blockin it returns liv
\$headers	<pre>= \$response-&gt;getHeaders();</pre>	gets the HTTP headers as string[][] with the header names lower-cased	response_headers info about debug the response
\$content	<pre>= \$response-&gt;getContent();</pre>	gets the response body as a string	url
\$httpInfo	= \$response-> <mark>getInfo()</mark> ;	gets info coming from the transport layer	error http_method
\$startTime	<pre>= \$response-&gt;getInfo('start_time');</pre>	-	http_code
			> redirect count

## Streaming Responses for full async apps

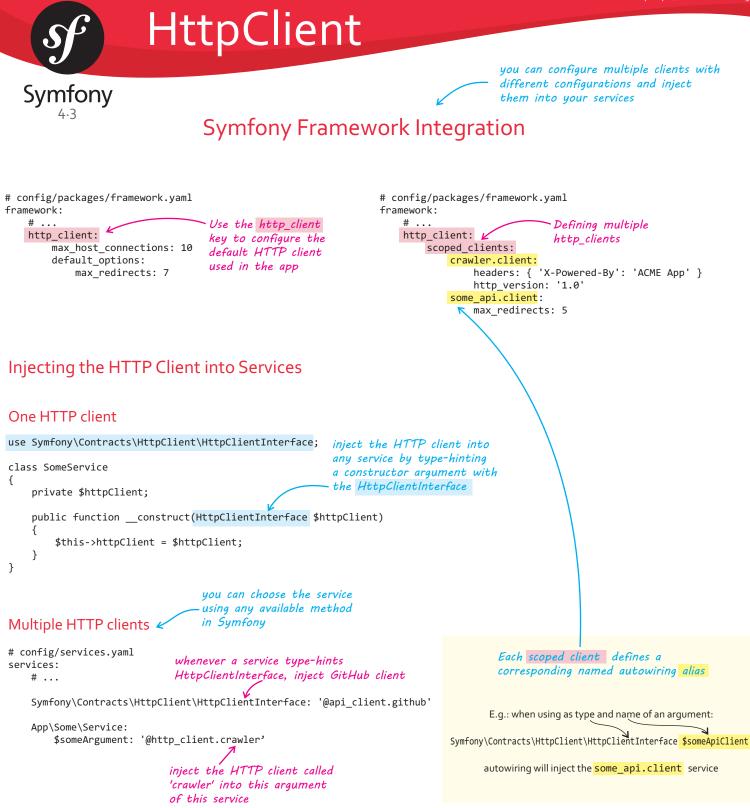


is non-blocking: ta it returns live e\_headers info about the response thod de redirect\_count start\_time connect time redirect\_time starttransfer time total\_time namelookup\_time size\_upload size\_download primary\_ip primary\_port redirect\_url gets detailed

logs about the HTTP transaction E.g.: \$response->getInfo('debug')

response chunks implement Symfony\Contracts\HttpClient\ChunkInterface





### 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:

```
// 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);