stripe / **stripe-php**

PHP library for the Stripe API. https://stripe.com

#stripe

925 commits	2 branches	161 releases	114 contributors	MIT

Branch: master ▾ | New pull request | Find file | Clone or download ▾

brandur Bump version to 6.6.0 — Latest commit `0bae101` 6 days ago

.github	Added a template for GitHub issues	2 years ago
data	Update CA bundle and add script for future updates	2 months ago
examples	Use short array syntax everywhere	2 months ago
lib	Bump version to 6.6.0	6 days ago
tests	flexible and metered billing support	6 days ago
.coveralls.yml	Add Support for coveralls code coverage	3 years ago
.gitignore	Update gitignore file	4 months ago
.travis.yml	Upgrade stripe-mock to 0.11.2	12 days ago
CHANGELOG.md	Bump version to 6.6.0	6 days ago
LICENSE	Update license copyright year.	3 years ago
README.md	Add README section for "per-request configuration"	2 months ago
VERSION	Bump version to 6.6.0	6 days ago
build.php	Drop support for PHP 5.3	2 months ago
composer.json	Drop support for PHP 5.3	2 months ago
init.php	flexible and metered billing support	6 days ago
phpunit.no_autoload.xml	Add init file	3 years ago
phpunit.xml	Remove forceCoversAnnotation from phpunit.xml	3 years ago
update_certs.php	Update CA bundle and add script for future updates	2 months ago

README.md

Stripe PHP bindings

build passing | stable v6.6.0 | downloads 6.87 M | license MIT | coverage 90%

You can sign up for a Stripe account at https://stripe.com.

Requirements

PHP 5.4.0 and later.

Composer

You can install the bindings via Composer. Run the following command:

```
composer require stripe/stripe-php
```

To use the bindings, use Composer's autoload:

```
require_once('vendor/autoload.php');
```

Manual Installation

If you do not wish to use Composer, you can download the latest release. Then, to use the bindings, include the `init.php` file.

```
require_once('/path/to/stripe-php/init.php');
```

Dependencies

The bindings require the following extensions in order to work properly:

- `curl`, although you can use your own non-cURL client if you prefer
- `json`
- `mbstring` (Multibyte String)

If you use Composer, these dependencies should be handled automatically. If you install manually, you'll want to make sure that these extensions are available.

Getting Started

Simple usage looks like:

```
\Stripe\Stripe::setApiKey('sk_test_BQokikJOvBiI2HlWgH4olfQ2');
$charge = \Stripe\Charge::create(['amount' => 2000, 'currency' => 'usd', 'source' => 'tok_189fqt2eZvKYlo2CTGBeg6Uq'])
echo $charge;
```

Documentation

Please see https://stripe.com/docs/api for up-to-date documentation.

Legacy Version Support

PHP 5.3

If you are using PHP 5.3, you can download v5.8.0 (zip, tar.gz) from our releases page. This version will continue to work with new versions of the Stripe API for all common uses.

PHP 5.2

If you are using PHP 5.2, you can download v1.18.0 (zip, tar.gz) from our releases page. This version will continue to work with new versions of the Stripe API for all common uses.

This legacy version may be included via `require_once("/path/to/stripe-php/lib/Stripe.php");`, and used like:

```
Stripe::setApiKey('d8e8fca2dc0f896fd7cb4cb0031ba249');
$charge = Stripe_Charge::create(array('source' => 'tok_XXXXXXXX', 'amount' => 2000, 'currency' => 'usd'));
echo $charge;
```

Custom Request Timeouts

NOTE: We do not recommend decreasing the timeout for non-read-only calls (e.g. charge creation), since even if you locally timeout, the request on Stripe's side can still complete. If you are decreasing timeouts on these calls, make sure to use idempotency tokens to avoid executing the same transaction twice as a result of timeout retry logic.

To modify request timeouts (connect or total, in seconds) you'll need to tell the API client to use a CurlClient other than its default. You'll set the timeouts in that CurlClient.

```
// set up your tweaked Curl client
$curl = new \Stripe\HttpClient\CurlClient();
$curl->setTimeout(10); // default is \Stripe\HttpClient\CurlClient::DEFAULT_TIMEOUT
$curl->setConnectTimeout(5); // default is \Stripe\HttpClient\CurlClient::DEFAULT_CONNECT_TIMEOUT

echo $curl->getTimeout(); // 10
echo $curl->getConnectTimeout(); // 5

// tell Stripe to use the tweaked client
\Stripe\ApiRequestor::setHttpClient($curl);

// use the Stripe API client as you normally would
```

Custom cURL Options (e.g. proxies)

Need to set a proxy for your requests? Pass in the requisite `CURLOPT_*` array to the CurlClient constructor, using the same syntax as `curl_stopt_array()`. This will set the default cURL options for each HTTP request made by the SDK, though many more common options (e.g. timeouts; see above on how to set those) will be overridden by the client even if set here.

```
// set up your tweaked Curl client
$curl = new \Stripe\HttpClient\CurlClient([CURLOPT_PROXY => 'proxy.local:80']);
// tell Stripe to use the tweaked client
\Stripe\ApiRequestor::setHttpClient($curl);
```

Alternately, a callable can be passed to the CurlClient constructor that returns the above array based on request inputs. See `testDefaultOptions()` in `tests/CurlClientTest.php` for an example of this behavior. Note that the callable is called at the beginning of every API request, before the request is sent.

Configuring a Logger

The library does minimal logging, but it can be configured with a `PSR-3` compatible logger so that messages end up there instead of `error_log`:

```
\Stripe\Stripe::setLogger($logger);
```

Accessing response data

You can access the data from the last API response on any object via `getLastResponse()`.

```
$charge = \Stripe\Charge::create(['amount' => 2000, 'currency' => 'usd', 'source' => 'tok_visa']);
echo $charge->getLastResponse()->headers['Request-Id'];
```

SSL / TLS compatibility issues

Stripe's API now requires that all connections use TLS 1.2. Some systems (most notably some older CentOS and RHEL versions) are capable of using TLS 1.2 but will use TLS 1.0 or 1.1 by default. In this case, you'd get an `invalid_request_error` with the following error message: "Stripe no longer supports API requests made with TLS 1.0. Please initiate HTTPS connections with TLS 1.2 or later. You can learn more about this at https://stripe.com/blog/upgrading-tls.".

The recommended course of action is to upgrade your cURL and OpenSSL packages so that TLS 1.2 is used by default, but if that is not possible, you might be able to solve the issue by setting the `CURLOPT_SSLVERSION` option to either `CURL_SSLVERSION_TLSv1` or `CURL_SSLVERSION_TLSv1_2`:

```
$curl = new \Stripe\HttpClient\CurlClient([CURLOPT_SSLVERSION => CURL_SSLVERSION_TLSv1]);
\Stripe\ApiRequestor::setHttpClient($curl);
```

Per-request Configuration

For apps that need to use multiple keys during the lifetime of a process, like one that uses Stripe Connect, it's also possible to set a per-request key and/or account:

```
\Stripe\Charge::all([], [
    'api_key' => 'sk_test_...',
    'stripe_account' => 'acct_...'
]);

\Stripe\Charge::retrieve("ch_18atAXCdGbJFKhCuBAa4532Z", [
    'api_key' => 'sk_test_...',
    'stripe_account' => 'acct_...'
]);
```

Configuring CA Bundles

By default, the library will use its own internal bundle of known CA certificates, but it's possible to configure your own:

```
\Stripe\Stripe::setCABundlePath("path/to/ca/bundle");
```

Configuring Automatic Retries

The library can be configured to automatically retry requests that fail due to an intermittent network problem:

```
\Stripe\Stripe::setMaxNetworkRetries(2);
```

Idempotency keys are added to requests to guarantee that retries are safe.

Development

Install dependencies:

```
composer install
```

The test suite depends on stripe-mock, so make sure to fetch and run it from a background terminal (stripe-mock's README also contains instructions for installing via Homebrew and other methods):

```
go get -u github.com/stripe/stripe-mock
stripe-mock
```

Install dependencies as mentioned above (which will resolve PHPUnit), then you can run the test suite:

```
./vendor/bin/phpunit
```

Or to run an individual test file:

```
./vendor/bin/phpunit tests/UtilTest.php
```

Update bundled CA certificates from the Mozilla cURL release:

```
./update_certs.php
```

Attention plugin developers

Are you writing a plugin that integrates Stripe and embeds our library? Then please use the `setAppInfo` function to identify your plugin. For example:

```
\Stripe\Stripe::setAppInfo("MyAwesomePlugin", "1.2.34", "https://myawesomeplugin.info");
```

The method should be called once, before any request is sent to the API. The second and third parameters are optional.

SSL / TLS configuration option

See the "SSL / TLS compatibility issues" paragraph above for full context. If you want to ensure that your plugin can be used on all systems, you should add a configuration option to let your users choose between different values for `CURLOPT_SSLVERSION`: none (default), `CURL_SSLVERSION_TLSv1` and `CURL_SSLVERSION_TLSv1_2`.