PostNL PHP bindings Documentation

Release 1.0.0

Read the Docs

May 16, 2018

Contents

These API bindings make it easy to connect to PostNL's CIF API, used for retrieving delivery options, printing shipment labels and finding shipment statuses.

- It has a simple interface for connecting with either the legacy, SOAP or REST API.
- Abstracts away direct requests to the API, allowing you to focus on the code itself. The object structure is based on the SOAP API.
- Can merge PDF labels (both A6 and A4) and automatically sends concurrent requests when necessary, making batch processing a lot easier easier.
- Follows PHP standards, some of them are:
  - PSR-7 interfaces for requests and responses. Build and process functions are provided so you can create your own mix of batch requests.
  - PSR-6 caching, so you can use your favorite cache for caching API responses.
  - PSR-3 logging. You can log the requests and responses for debugging purposes.
- Framework agnostic. You can use this library with any framework.
- A custom HTTP client interface so you can use the HTTP client of your choice. Using the Guzzle client is strongly recommended, because of its higher performance and superb error correction.

```php
<?php
$postnl = new PostNL(...);
$timeframes = $postnl->getTimeframes(
    (new GetTimeframes())
        ->setTimeframe([Timeframe::create([
            'CountryCode'   => 'NL',
            'StartDate'     => date('d-m-Y', strtotime('+1 day')),
            'EndDate'       => date('d-m-Y', strtotime('+14 days')),
            'HouseNr'       => 42,
            'PostalCode'    => '2132WT',
            'SundaySorting' => true,
            'Options'       => ['Daytime', 'Evening'],
        ])])
);
var_dump($timeframes);
```

CHAPTER 1

Developer Guide

1.1 Overview

1.1.1 Requirements

1. PHP 5.5.5 or higher
2. JSON extension
3. XML Support (SimpleXMLElement)
4. By default this library utilizes cURL for communication.
5. To use the cURL client, you must have a recent version of cURL >= 7.19.4 compiled with OpenSSL and zlib.

Note: If you use the Guzzle client, you do not need to have the cURL extension installed. As an alternative, you can enable `allow_url_fopen` in your system's php.ini. The included Guzzle version can work with the PHP stream wrapper to handle HTTP requests. For more information check out Guzzle's documentation.

1.1.2 Installation

The recommended way to install the PostNL library is with Composer. Composer is a dependency management tool for PHP that allows you to declare the dependencies your project needs and installs them into your project.

```
# Install Composer
curl -sS https://getcomposer.org/installer | php
```

Install the PostNL library:

```
php composer.phar require thirtybees/postnl-api-php:~1.0
```

You can optionally add Guzzle as a dependency using the composer.phar CLI:

```
php composer.phar require guzzlehttp/guzzle:~6.3
```

Alternatively, you can specify Guzzle as a dependency in your project's existing composer.json file:

```
{
   "require": {
      "guzzlehttp/guzzle": "~6.3"
   }
}
```

After installing, you need to require Composer's autoloader:

```
require 'vendor/autoload.php';
```

You can find out more on how to install Composer, configure autoloading, and other best-practices for defining dependencies at getcomposer.org.

Bleeding edge

During your development, you can keep up with the latest changes on the master branch by setting the version requirement for this library to `~1.0@dev`.

```
{
   "require": {
      "thirtybees/postnl-api-php": "~1.0@dev"
   }
}
```

1.1.3 License

Licensed using the MIT license.

> Copyright (c) 2017-2018 thirty bees <https://github.com/thirtybees>
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.1.4 Contributing

Guidelines

1. This library utilizes PSR-1, PSR-2, PSR-3, PSR-4, PSR-6 and PSR-7.
2. The library is meant to be lean, fast and sticks to the standards of the SOAP API. This means that not every feature request will be accepted.
3. The PostNL library has a minimum PHP version requirement of PHP 5.5.5. Pull requests must not require a PHP version greater than PHP 5.5.5.
4. All pull requests must include unit tests to ensure the change works as expected and to prevent regressions.

Running the tests

In order to contribute, you'll need to checkout the source from GitHub and install the dependencies using Composer:

```
git clone https://github.com/thirtybees/postnl-api-php.git
cd postnl-api-php && curl -s http://getcomposer.org/installer | php && ./composer.
↪phar install --dev
```

This library is unit tested with PHPUnit. Run the tests using the included PHPUnit version:

```
cd tests/
php ../vendor/bin/phpunit --testdox
```

Note: You'll need to use PHP 5.6 or newer in order to perform the tests.

1.2 Quickstart

This page provides a quick introduction to this library and a few examples. If you do not have the library installed, head over to the *Installation* page.

1.2.1 Making a Request

You can send requests by creating the request objects and passing them to one of the functions in the main `PostNL` class.

Requesting timeframes, locations and the delivery date

You can the timeframes, locations and delivery date at once to quickly retrieve all the available delivery options. Here's how it is done:

```
<?php
```

use ThirtyBeesPostNLEntityLabel; use ThirtyBeesPostNLPostNL; use ThirtyBeesPostNLEntityCustomer; use ThirtyBeesPostNLEntityAddress; use ThirtyBeesPostNLEntityShipment; use ThirtyBeesPostNLEntityDimension;

require_once __DIR__.'/vendor/autoload.php';

```
// Your PostNL credentials $customer = Customer::create([
    'CollectionLocation' => '123456', 'CustomerCode' => 'DEVC', 'CustomerNumber' =>
    '11223344', 'ContactPerson' => 'Lesley', 'Address' => Address::create([
        'AddressType' => '02', 'City' => 'Hoofddorp', 'CompanyName' => 'PostNL', 'Coun-
        trycode' => 'NL', 'HouseNr' => '42', 'Street' => 'Siriusdreef', 'Zipcode' => '2132WT',
    ]), 'Email' => 'michael@thirtybees.com', 'Name' => 'Michael',
]);
$apikey = 'YOUR_API_KEY_HERE'; $sandbox = true;
$postnl = new PostNL($customer, $apikey, $sandbox, PostNL::MODE_SOAP);
$mondayDelivery = true; $deliveryDaysWindow = 7; // Amount of days to show ahead $dropoffDelay = 0; //
Amount of days to delay delivery
$cutoffTime = '15:00:00'; $dropoffDays = [1 => true, 2 => true, 3 => true, 4 => true, 5 => true, 6 => false, 7
=> false]; foreach (range(1, 7) as $day) {
    if (isset($dropoffDays[$day])) {
        $cutOffTimes[] = new CutOffTime( str_pad($day, 2, '0', STR_PAD_LEFT), date('H:i:00',
            strtotime($cutoffTime)), true
        );
    }
}
$response = $postnl->getTimeframesAndNearestLocations(
    (new GetTimeframes())
        ->setTimeframe([
            (new Timeframe()) ->setCountryCode('NL') ->setEndDate(date('d-m-Y', str-
            totime('' +{$deliveryDaysWindow} days +{$dropoffDelay} days''))) -
            >setHouseNr('66') ->setOptions(['Morning', 'Daytime']) ->setPostalCode('2132WT')
            ->setStartDate(date('d-m-Y', strtotime('' +1 day +{$request['dropoff_delay']} days'')))
            ->setSundaySorting(!empty($mondayDelivery) && date('w', strtotime(''+{$dropoffDelay}
            days'')))
        ]),
    (new GetNearestLocations()) ->setCountrycode($request['cc']) ->setLocation(
            (new Location()) ->setAllowSundaySorting(!empty($mondayDelivery)) -
            >setDeliveryOptions(['PG', 'PGE']) ->setOptions(['Daytime']) ->setHouseNr('66')
            ->setPostalcode('2132WT')
        ),
    (new GetDeliveryDate())
        ->setGetDeliveryDate(
            (new GetDeliveryDate()) ->setAllowSundaySorting(!empty($mondayDelivery)) -
            >setCountryCode('NL') ->setCutOffTimes($cutOffTimes) ->setHouseNr($request['number'])
            ->setOptions($deliveryOptions) ->setPostalCode('2132WT') ->setShippingDate(date('d-m-Y
            H:i:s')) ->setShippingDuration(strval(1 + (int) $dropoffDelay))
        ) ->setMessage(new Message())
```

```
);
```

The response variable will contain the timeframes, nearest locations and delivery date. The reponse will be an array with the keys *timeframes*, *locations* and *delivery_date*. You can then use the delivery date to prune any timeframes that can no longer be guaranteed.

Requesting a merged label

Here is how you can request two labels and have them merged into a single PDF automatically:

```
<?php
use ThirtyBees\PostNL\Entity\Label;
use ThirtyBees\PostNL\PostNL;
use ThirtyBees\PostNL\Entity\Customer;
use ThirtyBees\PostNL\Entity\Address;
use ThirtyBees\PostNL\Entity\Shipment;
use ThirtyBees\PostNL\Entity\Dimension;

require_once __DIR__.'/vendor/autoload.php';

// Your PostNL credentials
$customer = Customer::create([
    'CollectionLocation' => '123456',
    'CustomerCode'       => 'DEVC',
    'CustomerNumber'     => '11223344',
    'ContactPerson'      => 'Lesley',
    'Address'            => Address::create([
        'AddressType' => '02',
        'City'        => 'Hoofddorp',
        'CompanyName' => 'PostNL',
        'Countrycode' => 'NL',
        'HouseNr'     => '42',
        'Street'      => 'Siriusdreef',
        'Zipcode'     => '2132WT',
    ]),
    'Email'              => 'michael@thirtybees.com',
    'Name'               => 'Michael',
]);

$apikey = 'YOUR_API_KEY_HERE';
$sandbox = true;

$postnl = new PostNL($customer, $apikey, $sandbox, PostNL::MODE_SOAP);

$barcodes = $postnl->generateBarcodesByCountryCodes(['NL' => 2]);

$shipments = [
    Shipment::create([
        'Addresses'          => [
            Address::create([
                'AddressType' => '01',
                'City'        => 'Utrecht',
                'Countrycode' => 'NL',
                'FirstName'   => 'Peter',
                'HouseNr'     => '9',
                'HouseNrExt'  => 'a bis',
                'Name'        => 'de Ruijter',
```

(continues on next page)

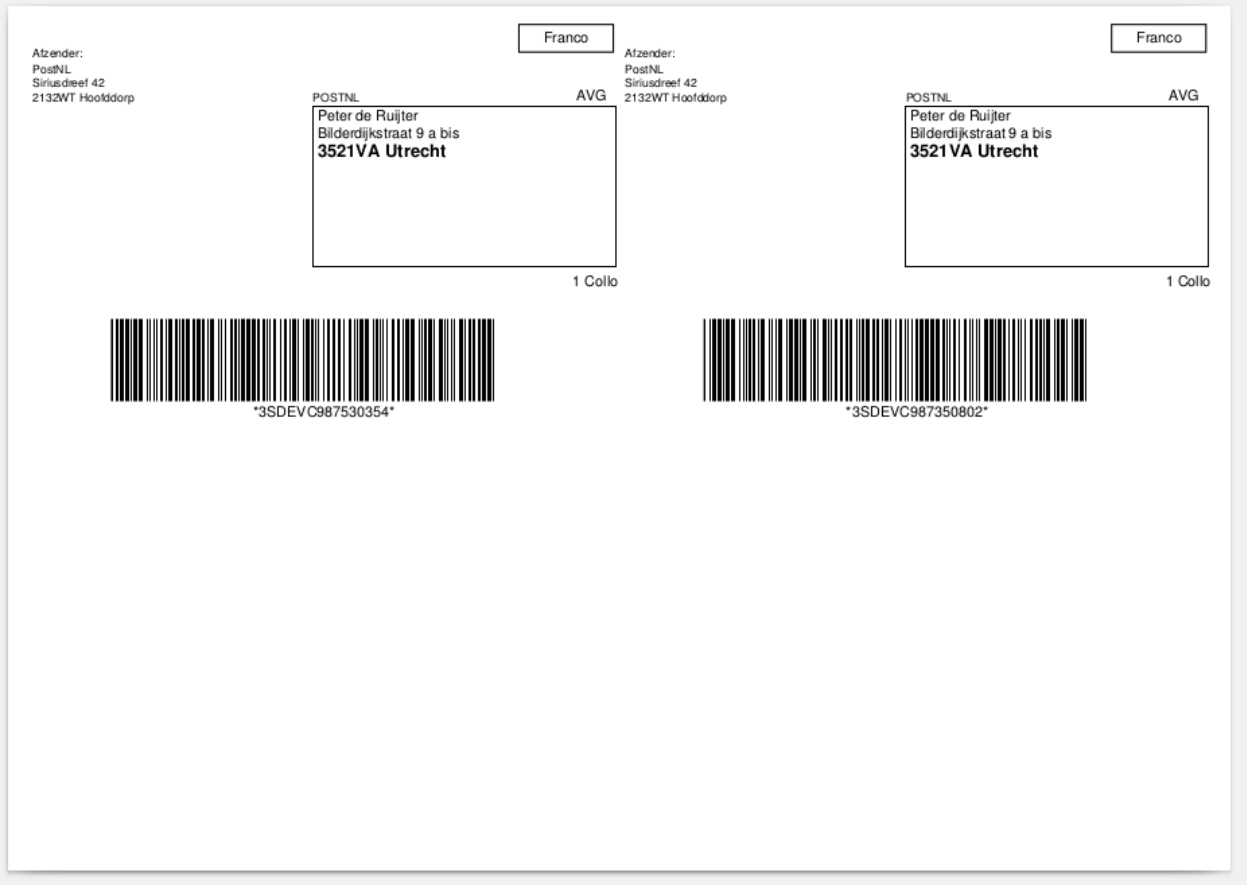
(continued from previous page)

```
                'Street'      => 'Bilderdijkstraat',
                'Zipcode'     => '3521VA',
            ]),
        ],
        'Barcode'             => $barcodes['NL'][0],
        'Dimension'           => new Dimension('1000'),
        'ProductCodeDelivery' => '3085',
    ]),
    Shipment::create([
        'Addresses'           => [
            Address::create([
                'AddressType' => '01',
                'City'        => 'Utrecht',
                'Countrycode' => 'NL',
                'FirstName'   => 'Peter',
                'HouseNr'     => '9',
                'HouseNrExt'  => 'a bis',
                'Name'        => 'de Ruijter',
                'Street'      => 'Bilderdijkstraat',
                'Zipcode'     => '3521VA',
            ]),
        ],
        'Barcode'             => $barcodes['NL'][1],
        'Dimension'           => new Dimension('1000'),
        'ProductCodeDelivery' => '3085',
    ]),
];

$label = $postnl->generateLabels(
    $shipments,
    'GraphicFile|PDF', // Printertype (only PDFs can be merged -- no need to use the␣
↪Merged types)
    true, // Confirm immediately
    true, // Merge
    Label::FORMAT_A4, // Format -- this merges multiple A6 labels onto an A4
    [
        1 => true,
        2 => true,
        3 => true,
        4 => true,
    ] // Positions
);

file_put_contents('labels.pdf', $label);
```

This will write a `labels.pdf` file that looks like this:



The PostNL client constructor accepts a few options:

customer `Customer` - *required*

The `Customer` object that is used to configure the client and let PostNL know who is requesting the data.

```
<?php
// Create a new customer
$client = new Customer::create([
  'CollectionLocation' => '123456',                    // Your collection␣
↪location
  'CustomerCode'       => 'DEVC',                      // Your Customer Code
  'CustomerNumber'     => '11223344',                  // Your Customer Number
  'GlobalPackBarcodeType('CX'),                        // Add your GlobalPack␣
↪information if you need
  'GlobalPackCustomerCode('1234'),                     // to create␣
↪international shipment labels
  'ContactPerson'      => 'Lesley',
  'Address'            => Address::create([
      'AddressType' => '02',                           // This address will be␣
↪shown on the labels
      'City'        => 'Hoofddorp',
      'CompanyName' => 'PostNL',
      'Countrycode' => 'NL',
      'HouseNr'     => '42',
      'Street'      => 'Siriusdreef',
      'Zipcode'     => '2132WT',
  ]),
```

(continues on next page)

(continued from previous page)

```
    'Email'              => 'michael@thirtybees.com',
    'Name'               => 'Michael',
]);
```

apikey `string``|``UsernameToken` - *required*

The `apikey` to use for the API. Note that if you want to switch from the legacy API to the new SOAP and REST API you will have to request a new key. The username can be omitted. If you want to connect to the legacy API you should pass a `UsernameToken` with your username and token set:

```
<?php
$usernameToken = new UsernameToken('username', 'token');
```

You can request an API key for the sandbox environment on this page: https://developer.postnl.nl/content/request-api-key For a live key you should contact your PostNL account manager.

sandbox `bool` - *required*

Indicate whether you'd like to connect to the sandbox environment. When *false* the library uses the live endpoints.

mode `int` - *optional, defaults to REST*

This library provides three ways to connect to the API:

- 1: REST mode
- 2: SOAP mode
- 5: Legacy mode – This is the previous SOAP API, which at the moment of writing is still in operation.

1.2.2 Building Requests

In most cases you would want to create request objects and pass them to one of the methods of the main object (`PostNL`). One exception is the Barcode Service. You can directly request multiple barcodes and for multiple countries at once. The library will internally handle the concurrent requests to the API.

In the above-mentioned merged label example we are passing two `Shipment` objects, filled with the needed information to generate the labels. To merge those labels manually, we have to set the merge option to `false` and can omit both the `format` and `positions` parameters. This will in turn make the library return `GenerateLabelResponse` objects.

These are in line with the `GenerateLabelResponse` nodes generated by the SOAP API, even when using the REST API. The main reason for this standardization is that the SOAP API has better documentation. If you need a quick reference of the `GenerateLabelResponse` object, you can either look up the code of the `GenerateLabelResponse` class or navigate to the API documentation directly.

Sending concurrent requests

There is no direct need to manually handle concurrent requests. This library handles most cases automatically and even provides a special function to quickly grab timeframe and location data for frontend delivery options widgets.

In case you manually want to send a custom mix of requests, you can look up the corresponding functions in the Service class of your choice and call the `` `buildXXXXXXRequest()` `` functions manually. Thanks to the PSR-7 standard used by this library you can use the `Request` object that is returned to access the full request that would otherwise be sent directly. To pick up where you left off you can then grab the response and pass it to one of the

`processXXXXXXXResponse()` functions of the Service class. The easiest method is to grab the raw HTTP message and parse it with the included PSR-7 library. An example can be found in the cURL client.

1.2.3 Using Response objects

Note: This section refers to Response objects returned by the library, not the standardized PSR-7 messages.

As soon as you've done your first request with this library, you will find that it returns a Response object. As mentioned in the *Building Requests* section, these Response objects are based on the SOAP API, regardless of the mode set. The properties of a Response object can be looked up in the code, but it can be a bit confusing at times, since the Response object will likely not contain all properties at once. It often depends on the context of the request. For this reason, you're better off by having a look at the SOAP API documentation directly or by checking out some of the examples in this documentation.

1.3 Services

1.3.1 Barcode service

Note:

PostNL API documentation for this service:
https://developer.postnl.nl/apis/barcode-webservice/overview

The barcode service allows you to generate barcodes for your shipment labels. Usually you would reserve an amount of barcodes, generate shipping labels and eventually confirm those labels. According to PostNL, this flow is necessary for a higher delivery success rate.

Generate a single barcode

You can generate a single barcode for domestic shipments as follows:

```
<?php
$postnl->generateBarcode();
```

This will generate a 3S barcode meant for domestic shipments only.

The function accepts the following arguments:

type `string` - *optional, defaults to 3S*

The barcode type. This is 2S/3S for the Netherlands and EU Pack Special shipments. For other destinations this is your GlobalPack barcode type. For more info, check the PostNL barcode service page.

range `string` - *optional, can be found automatically*

For domestic and EU shipments this is your customer code. Otherwise, your GlobalPack customer code.

serie `string` - *optional, can be found automatically*

This is the barcode range for your shipment(s). Check the PostNL barcode service page for the ranges that are available.

eps `bool` - *optional, defaults to false*

Indicates whether this is an EU Pack Special shipment.

Generate a barcode by country code

It is possible to generate a barcode by country code. This will let the library figure out what type, range, serie to use.

Example:

```
<?php
$postnl->generateBarcodeByCountryCode('BE');
```

This will generate a 3S barcode meant for domestic shipments only.

The function accepts the following arguments:

iso `string` - *required*

The two letter country ISO code. Make sure you use UPPERCASE.

Generate multiple barcodes by using country codes

You can generate a whole batch of barcodes at once by providing country codes and the amounts you would like to generate.

Example:

```
<?php
$postnl->generateBarcodeByCountryCode(['NL' => 2, 'DE' => 3]);
```

This will return a list of barcodes:

```
<?php
[
    'NL' => [
        '3SDEVC11111111111',
        '3SDEVC22222222222',
    ],
    'DE' => [
        '3SDEVC111111111',
        '3SDEVC222222222',
        '3SDEVC333333333',
    ],
];
```

The function accepts the following argument:

type `string` - *required*

An associative array with country codes as key and the amount of barcodes you'd like to generate per country as the value.

1.3.2 Labelling service

Note:

PostNL API documentation for this service:
https://developer.postnl.nl/apis/labelling-webservice

The labelling service allows you to create shipment labels and optionally confirm the shipments. The library has a built-in way to merge labels automatically, so you can request labels for multiple shipments at once.

Generate a single label

The following example generates a single shipment label for a domestic shipment:

```php
<?php
$postnl = new PostNL(...);
$postnl->generateLabel(
    Shipment::create()
        ->setAddresses([
            Address::create([
                'AddressType' => '01',
                'City'        => 'Utrecht',
                'Countrycode' => 'NL',
                'FirstName'   => 'Peter',
                'HouseNr'     => '9',
                'HouseNrExt'  => 'a bis',
                'Name'        => 'de Ruijter',
                'Street'      => 'Bilderdijkstraat',
                'Zipcode'     => '3521VA',
            ]),
            Address::create([
                'AddressType' => '02',
                'City'        => 'Hoofddorp',
                'CompanyName' => 'PostNL',
                'Countrycode' => 'NL',
                'HouseNr'     => '42',
                'Street'      => 'Siriusdreef',
                'Zipcode'     => '2132WT',
            ]),
        ])
        ->setBarcode($barcode)
        ->setDeliveryAddress('01')
        ->setDimension(new Dimension('2000'))
        ->setProductCodeDelivery('3085'),
    'GraphicFile|PDF',
    false
);
```

This will create a standard shipment (product code 3085). You can access the label (base64 encoded PDF) this way:

```php
<?php
$pdf = base64_decode($label->getResponseShipments()[0]->getLabels()[0]->getContent());
```

This function accepts the following arguments:

shipment `Shipment` - *required*

The Shipment object. Visit the PostNL API documentation to find out what a Shipment object consists of.

printertype `string` - *optional, defaults to GraphicFile|PDF*

The list of supported printer types can be found on this page: https://developer.postnl.nl/browse-apis/send-and-track/labelling-webservice/documentation-soap/

confirm `string` - *optional, defaults to true*

Indicates whether the shipment should immediately be confirmed.

Generate multiple shipment labels

The following example shows how a label can be merged:

```php
<?php
$shipments = [
    Shipment::create([
        'Addresses'           => [
            Address::create([
                'AddressType' => '01',
                'City'        => 'Utrecht',
                'Countrycode' => 'NL',
                'FirstName'   => 'Peter',
                'HouseNr'     => '9',
                'HouseNrExt'  => 'a bis',
                'Name'        => 'de Ruijter',
                'Street'      => 'Bilderdijkstraat',
                'Zipcode'     => '3521VA',
            ]),
        ],
        'Barcode'             => $barcodes['NL'][0],
        'Dimension'           => new Dimension('1000'),
        'ProductCodeDelivery' => '3085',
    ]),
    Shipment::create([
        'Addresses'           => [
            Address::create([
                'AddressType' => '01',
                'City'        => 'Utrecht',
                'Countrycode' => 'NL',
                'FirstName'   => 'Peter',
                'HouseNr'     => '9',
                'HouseNrExt'  => 'a bis',
                'Name'        => 'de Ruijter',
                'Street'      => 'Bilderdijkstraat',
                'Zipcode'     => '3521VA',
            ]),
        ],
        'Barcode'             => $barcodes['NL'][1],
        'Dimension'           => new Dimension('1000'),
        'ProductCodeDelivery' => '3085',
    ]),
];

$label = $postnl->generateLabels(
    $shipments,
    'GraphicFile|PDF', // Printertype (only PDFs can be merged -- no need to use the␣
↪Merged types)
    true, // Confirm immediately
    true, // Merge
    Label::FORMAT_A4, // Format -- this merges multiple A6 labels onto an A4
```

(continues on next page)

(continued from previous page)

```
    [
        1 => true,
        2 => true,
        3 => true,
        4 => true,
    ] // Positions
);

file_put_contents('labels.pdf', $label);
```

By setting the *merge* flag it will automatically merge the labels into a PDF string.

The function accepts the following arguments:

shipments `Shipment[]` - *required*

The Shipment objects. Visit the PostNL API documentation to find out what a Shipment object consists of.

printertype `string` - *optional, defaults to GraphicFile|PDF*

The list of supported printer types can be found on this page: https://developer.postnl.nl/browse-apis/send-and-track/labelling-webservice/documentation-soap/

confirm `string` - *optional, defaults to true*

Indicates whether the shipment should immediately be confirmed.

merge `bool` - *optional, default to false*

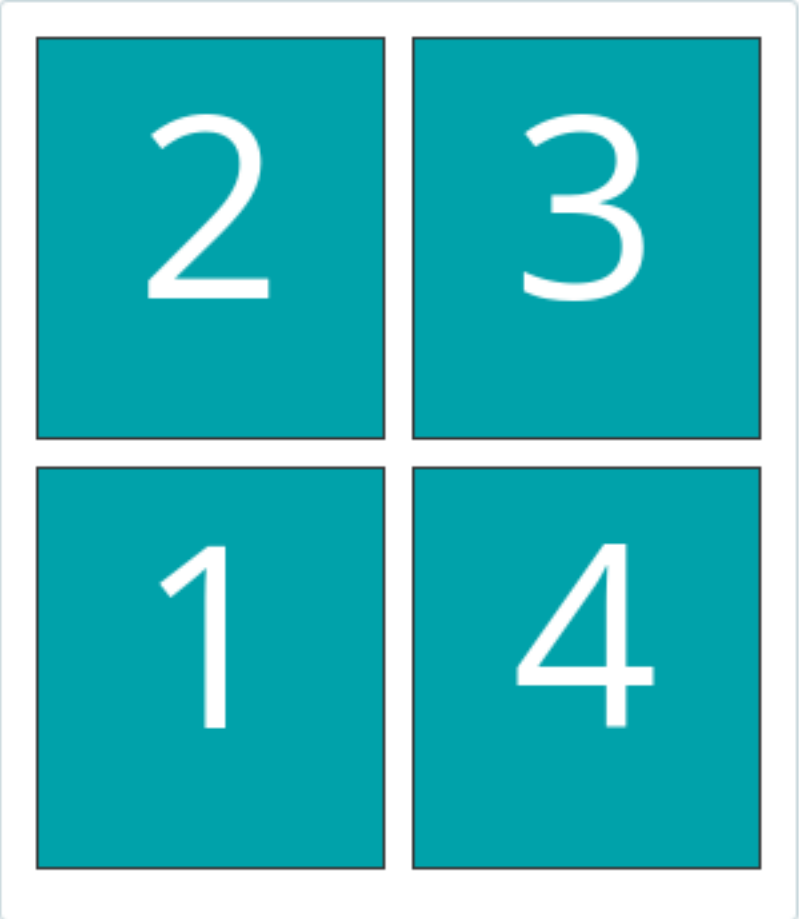
This will merge the labels and make the function return a pdf string of the merged label.

format `int` - *optional, defaults to 1 (FORMAT_A4)*

This sets the paper format (can be A4 or A4).

positions `bool[]` - *optional, defaults to all positions*

This will set the positions of the labels. The following image shows the available positions, use *true* or *false* to resp. enable or disable a position:



1.3.3 Confirming service

Note:

PostNL API documentation for this service:
https://developer.postnl.nl/apis/confirming-webservice

You can confirm shipments that have previously not been confirmed. The available methods are *confirmShipment* and *confirmShipments*. The first method accepts a single *Shipment* object whereas the latter accepts an array of 'Shipment's. The output is a boolean, or an array with booleans in case you are confirming multiple shipments. The results will be tied to the keys of your request array.

1.3.4 Shipping status service

Note:

PostNL API documentation for this service:
https://developer.postnl.nl/apis/shippingstatus-webservice

This service can be used to retrieve shipping statuses. For a short update use the *CurrentStatus* method, otherwise *CompleteStatus* will provide you with a long list containing the shipment's history.

Current Status by Barcode

Gets the current status by Barcode

```
<?php
 $postnl = new PostNL(...);
 $postnl->getCurrentStatus((new CurrentStatus())
     ->setShipment(
         (new Shipment())
             ->setBarcode('3SDEVC98237423')
     )
 );
```

statusrequest `CurrentStatus` - *required*

The CurrentStatus object. Check the API documentation for all possibilities.

Current Status by Reference

Gets the current status by reference. Note that you must have set the reference on the shipment label first.

```
<?php
 $postnl = new PostNL(...);
 $postnl->getCurrentStatusByReference((new CurrentStatusByReference())
     ->setShipment(
         (new Shipment())
             ->setReference('myref')
     )
 );
```

statusrequest `CurrentStatusByReference` - *required*

The CurrentStatusByReference object. Check the API documentation for all possibilities.

Current Status by Status Code

Gets the current status by status.

```
<?php
$postnl = new PostNL(...);
$postnl->getCurrentStatusByStatus((new CurrentStatusByStatus())
    ->setShipment(
        (new Shipment())
            ->setStatusCode('5')
    )
);
```

statusrequest `CurrentStatusByStatus` - *required*

The CurrentStatusByStatus object. Check the API documentation for all possibilities.

Current Status by Phase Code

Gets the current status by phase code. Note that the date range is required.

```
<?php
$postnl = new PostNL(...);
$postnl->getCurrentStatusByReference((new CurrentStatusByPhase())
    ->setShipment(
        (new Shipment())
            ->setPhaseCode('5')
            ->setDateFrom(date('d-m-Y H:i:s', strtotime('-7 days'))
            ->setDateTo(date('d-m-Y H:i:s')
    )
);
```

statusrequest `CurrentStatusByPhase` - *required*

The CurrentStatusByPhase object. Check the API documentation for all possibilities.

Complete Status by Barcode

Gets the complete status by Barcode

```
<?php
$postnl = new PostNL(...);
$postnl->getCompleteStatus((new CompleteStatus())
    ->setShipment(
        (new Shipment())
            ->setBarcode('3SDEVC98237423')
    )
);
```

statusrequest `CompleteStatus` - *required*

The CompleteStatus object. Check the API documentation for all possibilities.

Complete Status by Reference

Gets the complete status by reference. Note that you must have set the reference on the shipment label first.

```
<?php
$postnl = new PostNL(...);
$postnl->getCompleteStatusByReference((new CompleteStatusByReference())
    ->setShipment(
        (new Shipment())
            ->setReference('myref')
    )
);
```

statusrequest `CompleteStatusByReference` - *required*

The CompleteStatusByReference object. Check the API documentation for all possibilities.

Complete Status by Status Code

Gets the complete status by status.

```php
<?php
$postnl = new PostNL(...);
$postnl->getCompleteStatusByStatus((new CompleteStatusByStatus())
    ->setShipment(
        (new Shipment())
            ->setStatusCode('5')
    )
);
```

statusrequest `CompleteStatusByStatus` - *required*

The CompleteStatusByStatus object. Check the API documentation for all possibilities.

Complete Status by Phase Code

Gets the complete status by phase code. Note that the date range is required.

```php
<?php
$postnl = new PostNL(...);
$postnl->getCompleteStatusByReference((new CompleteStatusByPhase())
    ->setShipment(
        (new Shipment())
            ->setPhaseCode('5')
            ->setDateFrom(date('d-m-Y H:i:s', strtotime('-7 days'))
            ->setDateTo(date('d-m-Y H:i:s')
    )
);
```

statusrequest `CompleteStatusByPhase` - *required*

The CompleteStatusByPhase object. Check the API documentation for all possibilities.

Get Signature

Gets the signature of the shipment when available. A signature can be accessed by barcode only.

It accepts the following arguments

getsignature `GetSignature` - *required*

The *GetSignature* object. It needs to have one *Shipment* set with a barcode.

1.3.5 Delivery date service

Note:

PostNL API documentation for this service:
https://developer.postnl.nl/apis/deliverydate-webservice

Use the delivery date webservice to determine the delivery and shipping date. You can use this service to calculate the dates 'live' and to make sure you do not promise your customers any timeframes that are no longer available.

Get the Delivery Date

Here's how you can retrieve the closest delivery date:

```
<?php

$cutoffTime = '15:00:00';
$dropoffDays = [1 => true, 2 => true, 3 => true, 4 => true, 5 => true, 6 => false, 7
↪=> false];
foreach (range(1, 7) as $day) {
    if (isset($dropoffDays[$day])) {
        $cutOffTimes[] = new CutOffTime(
            str_pad($day, 2, '0', STR_PAD_LEFT),
            date('H:i:00', strtotime($cutoffTime)),
            true
        );
    }
}
$deliveryDate = $postnl->getDeliveryDate(
    (new GetDeliveryDate())
        ->setGetDeliveryDate(
            (new GetDeliveryDate())
                ->setAllowSundaySorting(false)
                ->setCountryCode('NL')
                ->setCutOffTimes($cutOffTimes)
                ->setHouseNr('66')
                ->setOptions(['Morning', 'Daytime'])
                ->setPostalCode('2132WT')
                ->setShippingDate(date('d-m-Y H:i:s'))
                ->setShippingDuration('1')
        )
);
```

The result will be a *GetDeliveryDateResponse*. Calling *getDeliveryDate* on this object will return the delivery date as a string in the *d-m-Y H:i:s* PHP date format.

The function accepts the following arguments

deliverydaterequest `GetDeliveryDate` - *required*

> The *GetDeliveryDate* request. See the API documentation for the possibilities. As shown in the example you will need to provide as many details as possible to get accurate availability information.

Get the Shipping Date

The Shipping Date service almost works in the same way as the Delivery Date service, except this time you provide the actual delivery date in order to calculate the closest shipping date.

```
<?php

$cutoffTime = '15:00:00';
$dropoffDays = [1 => true, 2 => true, 3 => true, 4 => true, 5 => true, 6 => false, 7
↪=> false];
foreach (range(1, 7) as $day) {
    if (isset($dropoffDays[$day])) {
        $cutOffTimes[] = new CutOffTime(
            str_pad($day, 2, '0', STR_PAD_LEFT),
```

(continues on next page)

(continued from previous page)

```
            date('H:i:00', strtotime($cutoffTime)),
            true
        );
    }
}
$deliveryDate = $postnl->getDeliveryDate(
    (new GetDeliveryDate())
        ->setGetDeliveryDate(
            (new GetDeliveryDate())
                ->setAllowSundaySorting(false)
                ->setCountryCode('NL')
                ->setCutOffTimes($cutOffTimes)
                ->setHouseNr('66')
                ->setOptions(['Morning', 'Daytime'])
                ->setPostalCode('2132WT')
                ->setShippingDate(date('d-m-Y H:i:s'))
                ->setShippingDuration('1')
        )
);
```

The function accepts the following arguments

shippingdaterequest `GetSentDate` - *required*

The *GetSentDate* request. See the API documentation for the possibilities. As shown in the example you will need to provide as many details as possible to get accurate availability information.

1.3.6 Timeframe service

Note:

PostNL API documentation for this service:
https://developer.postnl.nl/apis/timeframe-webservice

---

timeframes `GetTimeframes` - *required*

The *GetTimeframes* request object. See the API documentation for more details.

1.3.7 Location service

Note:

PostNL API documentation for this service:
https://developer.postnl.nl/apis/location-webservice

The location service allows you to retrieve a list of locations for the given postcode or coordinates.

Get Nearest Locations

Here's an example of how you can retrieve the nearest location by postcode:

```php
<?php
$postnl->getNearestLocations(
    (new GetNearestLocations())
        ->setCountrycode('NL')
        ->setLocation(
            (new Location())
                ->setAllowSundaySorting(false)
                ->setDeliveryOptions(['PG'])
                ->setOptions(['Daytime'])
                ->setHouseNr('66')
                ->setPostalcode('2132WT')
        )
    );
```

nearestlocations `GetNearestLocations` - *required*

The *GetNearestLocations* request object. See the API documentation for more details.

Get Nearest Locations by Coordinates

You can also get the locations by specifying a bounding box. One can be drawn by providing the North-West and South-East corner of the box:

```php
<?php

 $postnl->getLocationsInArea(
     (new GetLocationsInArea())
         ->setCountrycode('NL')
         ->setLocation(
             (new Location())
                 ->setAllowSundaySorting(false)
                 ->setDeliveryDate(date('d-m-Y', strtotime('+1 day')))
                 ->setDeliveryOptions([
                     'PG',
                 ])
                 ->setOptions([
                     'Daytime',
                 ])
                 ->setCoordinatesNorthWest(
                     (new CoordinatesNorthWest())
                         ->setLatitude((string) 52.156439)
                         ->setLongitude((string) 5.015643)
                 )
                 ->setCoordinatesSouthEast(
                     (new CoordinatesNorthWest())
                         ->setLatitude((string) 52.017473)
                         ->setLongitude((string) 5.065254)
                 )
         )
 );
```

This function accepts the arguments:

locationsinarea `GetLocationsInArea` - *required*

The *GetLocationsInArea* request object. See the API documentation for more details.

1.4 HTTP Client

By default the library will use cURL or Guzzle when available. You can always switch HTTP clients as follows:

```
<?php
$postnl = new PostNL(...);
$postnl->setHttpClient(\ThirtyBees\PostNL\HttpClient\CurlClient::getInstance());
```

You can create a custom HTTP Client by implementing the `\ThirtyBees\PostNL\HttpClient\ClientInterface` interface.

1.5 Caching

PSR-6 caching is supported, which means you can grab any caching library for PHP that you like and plug it right into this library.

Note that not all services can be cached. At the moment cacheable services are:

- Labelling webservice
- Timeframes webservice
- Location webservice
- Deliverydate webservice
- Shippingstatus webservice

To enable caching for a certain service you can use the following:

```
<?php
use Cache\Adapter\Filesystem\FilesystemCachePool;
use League\Flysystem\Adapter\Local;
use League\Flysystem\Filesystem;

// Cache in the `/cache` folder relative to this directory
$filesystemAdapter = new Local(__DIR__.'/');
$filesystem = new Filesystem($filesystemAdapter);

$postnl = new PostNL(...);

$labellingService = $postnl->getLabellingService();
$labellingService->cache = new FilesystemCachePool($filesystem);

// Set a TTL of 600 seconds
$labellingService->ttl = 600;

// Using a DateInterval (600 seconds)
$labellingServiceervice->ttl = new DateInterval('PT600S');

// Setting a deadline instead, useful for the timeframe service, so you can cache␣
↪until the cut-off-time or
// until the next day
$labellingServiceervice = $postnl->getTimeframeService();
$labellingService->ttl = new DateTime('14:00:00');
```

Note: This example used the Flysystem (filesystem) cache. An extensive list of supported caches can be found on this page.

1.6 Logging

Requests and responses can be logged for debugging purposes. In order to enable logging you will need to pass a PSR-3 compatible logger.

```
<?php
use League\Flysystem\Adapter\Local;
use League\Flysystem\Filesystem;

use Psr\Log\LogLevel;
use wappr\Logger;

// Initialize the file system adapter
$logfs = new Filesystem($adapter);

// Set the DEBUG log level
$logger = new Logger($logfs, LogLevel::DEBUG);

// Set the filename format, we're creating one file for every minute of request/
↪responses
$logger->setFilenameFormat('Y-m-d H:i');

// Set this logger for all services at once
$postnl->setLogger($logger);

// Set the logger for just the Labelling service
$postnl->getLabellingService()->setLogger($logger);
```

Note: This example used the Wappr logger. You can use any logger you like, as long as it implements the PSR-3 standard. The log level needs to be set at `DEBUG`.

1.7 Authors

- Michael Dekker <michael@thirtybees.com> (maintainer)
- niccifor (contributor)