Payments Hub Documentation

Sourcefabric

Jul 23, 2019

Contents:

CHAPTER 1

What is Payments Hub?

Payments Hub is a free and open-source framework for voluntary payment integration into different publishing channels based on Symfony Framework.

Note: This documentation assumes you have already a working knowledge of the Symfony Framework. If you're not familiar with Symfony, please start with reading the Quick Tour from the Symfony documentation.

CHAPTER 2

Features

- Pay using Paypal Express Checkout, Stripe Checkout, Credit Card, SEPA Direct Debit, Mbe4 phone bill, offline.
- Recurring & non-recurring subscriptions
- Manage and add custom payment methods
- Customization of the templates
- Possibility to change the application flows (redirect URLs)
- Push details of the subscription to 3rd party applications using webhooks
- API-centric architecture

CHAPTER 3

Getting Started

3.1 Setup

3.1.1 Installing Composer

Composer is the package manager used by modern PHP applications. Use Composer to manage dependencies in your Symfony applications and to install Symfony Components in your PHP projects.

Since this article was first published, Composer installation has improved a lot. Therefore, the original contents of this article have been removed and you are encouraged to install Composer as explained in the official Composer documentation.

Installation:

3.2 Installation

3.2.1 Prerequisites

This project requires the OpenSSL library to be installed and PHP >= 7.1.

3.2.2 Requirements

Firstly, check Symfony Requirements.

Operating Systems

- Linux or MacOS

Web Servers

Payments Hub can run on Nginx or Apache servers.

For the development purposes we recommend using Symfony's PHP's built-in web server.

See Configuring a Web Server section for more details.

Databases

- PostgreSQL 9.x
- MySQL 5.x

By default, the PostgreSQL (`pdo_pgsql`) driver is enabled.

Tip: To use MySQL (`pdo_mysql` driver), just simply set the value of `database_driver` parameter to `pdo_mysql` during the *installation* or set it manually in `app/config/parameters.yml`.

3.2.3 Installation

If you don't have Composer installed in your computer, start by *installing Composer globally*.

Clone the repository and install vendors using Composer:

Note: Before installing dependencies, make sure the *SSH keys needed for the API authentication are generated*.

Then run the following command:

```
composer install
```

When all dependencies are installed, you will be asked to fill the `parameters.yml` file which needs to be completed before continuing.

Then create the database schema:

Note: If the database does not exist, create it manually or by running command: `bin/console doctrine:database:create`.

Run:

```
bin/console doctrine:migrations:migrate
```

and start the Payments Hub using built-in PHP server:

```
bin/console server:start
```

Next, open `http://127.0.0.1:8000` in your browser and you will see Payments Hub running in the development mode.

For the production, you will need to configure the vhost using Nginx or Apache and run the project there, instead of using the built-in PHP server.

3.2.4 Configuration

Generate the SSH keys to properly use the API authentication:

```
mkdir var/jwt
openssl genrsa -out var/jwt/private.pem -aes256 4096
openssl rsa -pubout -in var/jwt/private.pem -out var/jwt/public.pem
```

The generated keys will be needed during the *installation* where you will need to fill the `parameters.yml` file configuration.

- *Prerequisites*
- *Requirements*
- *Installation*
- *Configuration*
- *Prerequisites*
- *Requirements*
- *Installation*
- *Configuration*

Subscriptions:

3.3 Subscriptions

3.3.1 Subscription's intervals

By default, there are three intervals configured as listed below:

- Monthly (1 month)
- Quarterly (3 months)
- Yearly (1 year)

Intervals are only used when the subscription of type `recurring` is used. For `non-recurring` subscription this field is ignored.

Why to change the subscription's intervals?

There might be a reason to customize the subscription's intervals to your own needs in order to provide different options to end users.

How to change the subscription's intervals?

The default intervals are configured in the `app/config/parameters.yml` file as an array under the `subscription_intervals` parameter:

```
# app/config/parameters.yml
parameters:
    # ..
    subscription_intervals:
        Monthly: 1 month
        Quarterly: 3 months
        Yearly: 1 year
```

To change the intervals, simply change the value of the `subscription_intervals` parameter inside `app/config/parameters.yml` file:

```
# app/config/parameters.yml
parameters:
    # ..
    subscription_intervals:
        Monthly: 1 month
        Quarterly: 3 months
        Half-yearly: 6 months
        Yearly: 1 year
```

Note that this is a key-value array where key is a label and the value is the real interval value that is used by the payment gateway. The value can be either: 1 month, 2 months, 4 months, 2 years etc.

That's it! New intervals will be now available in the system.

- *Subscription's intervals*
- *Subscription's intervals*

Payments:

3.4 Payments

3.4.1 How to support a new payment gateway?

Payments Hub is using Payum library to handle the payments.

There might be a reason to introduce a new payment gateway to support payments using different channels (e.g. phone bill, credit card etc.). Payum supports currently a lot of open-source gateways which list can be found here.

For a purpose of this guide, the Mbe4 Payum extension has been used.

1. Register gateway as a service

First, the gateway needs to be registered as a service inside your *services.yml* file so put the following definition into that file:

```
ph.payum.mbe4.factory:
    class: Payum\Core\Bridge\Symfony\Builder\GatewayFactoryBuilder
    arguments: [PayHelper\Payum\Mbe4\Mbe4GatewayFactory]
    tags:
        - { name: payum.gateway_factory_builder, factory: mbe4 }
```

Payments Hub will automatically register a new gateway in the registry/container so it can be available in the system.

2. A new gateway configuration form type

A form type will need to be created in order to be able to expose the gateway-specific configuration to the API and be able to save gateway's configuration in the database.

The configuration of every gateway should be documented in the Payum extension's documentation that is going to be used.

```php
<?php

declare(strict_types=1);

namespace PH\Bundle\PayumBundle\Form\Type;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\IntegerType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\Validator\Constraints\NotBlank;
use Symfony\Component\Validator\Constraints\Range;

final class Mbe4GatewayConfigurationType extends AbstractType
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('username', TextType::class, [
                'constraints' => [
                    new NotBlank([
                        'groups' => 'ph',
                    ]),
                ],
            ])
            ->add('password', TextType::class, [
                'constraints' => [
                    new NotBlank([
                        'groups' => 'ph',
                    ]),
                ],
            ])
            ->add('clientId', TextType::class, [
                'constraints' => [
                    new NotBlank([
                        'groups' => 'ph',
                    ]),
                ],
            ])
            ->add('serviceId', TextType::class, [
                'constraints' => [
                    new NotBlank([
                        'groups' => 'ph',
                    ]),
                ],
            ])
            ->add('contentclass', IntegerType::class, [
```

(continues on next page)

(continued from previous page)

```
                'constraints' => [
                    new NotBlank([
                        'groups' => 'ph',
                    ]),
                ],
            ])
            ->add('minAmount', IntegerType::class, [
                'constraints' => [
                    new NotBlank([
                        'groups' => 'ph',
                    ]),
                    new Range([
                        'min' => 0,
                        'groups' => 'ph',
                    ]),
                ],
            ])
            ->add('maxAmount', IntegerType::class, [
                'constraints' => [
                    new NotBlank([
                        'groups' => 'ph',
                    ]),
                ],
            ])
        ;
    }
}
```

Register a new form type as a service in `services.yml` file:

```
ph.form.type.gateway_configuration.mbe4:
    class: PH\Bundle\PayumBundle\Form\Type\Mbe4GatewayConfigurationType
    tags:
        - { name: sylius.gateway_configuration_type, type: 'mbe4', label: 'Mbe4' }
        - { name: form.type }
```

Note that the `sylius.gateway_configuration_type` tag has been added, which will add the configuration defined in the form type to the Payum gateway's configuration automatically so there is no need to define the configuration of the Payum gateway in the `config.yml` file.

That's it! A support for a new payment gateway is now added to the system and the Mbe4 gateway can be managed using API.

3.4.2 Supported Gateways

Currently, there are five payment gateways configured:

- Mollie
- Mbe4
- PayPal Express Checkout
- Stripe Checkout
- Offline
- *How to support a new payment gateway?*

- *Supported Gateways*
- *How to support a new payment gateway?*
- *Supported Gateways*

Custom Redirect URLs:

3.5 Custom Redirect URLs

Payments Hub allows to change the redirect URLs that are used by the payment gateways. By defaut, there are two main redirect URLs:

- thank you URL
- cancel URL

Once a user chooses the payment method and pays for the subscription, Payment Hub communicates with the remote gateway. Next, the remote gateway redirects back a user to the Payments Hub and then the internal redirect to the "thank you" URL is done. In case of the successful or failed payment, a user is redirected to the "thank you" URL. In case of cancellation, a user is redirected to the "cancel" URL.

These URLs can be changed so, for example, a user can be redirected to a different view on a different server.

3.5.1 How To Change Thank You URL?

The "thank you" URL can be changed by simply setting a `ph_payum_redirect_thank_you_url` parameter in `app/config/parameters.yml` file. The default value is `/thank-you` which is defined by the Payments Hub.

```
# app/config/parameters.yml
parameters:
    # ..
    ph_payum_redirect_thank_you_url: '/custom/thanks'
```

That's it! Now, if the payment was completed successfully or it failed, a user will be redirected to a newly configured URL.

3.5.2 How To Change Cancel URL?

The "cancel" URL can be changed by simply setting a `ph_payum_redirect_cancel_url` parameter in `app/config/parameters.yml` file. The default value is `/cancel` which is defined by the Payments Hub.

```
# app/config/parameters.yml
parameters:
    # ..
    ph_payum_redirect_cancel_url: '/custom/cancel'
```

That's it! Now, if the payment was cancelled, a user will be redirected to a newly configured URL.

- *How To Change Thank You URL?*
- *How To Change Cancel URL?*
- *How To Change Thank You URL?*
- *How To Change Cancel URL?*

Overriding templates:

3.6 Overriding templates

There are several templates which can be customized to your needs.

Available templates:

3.6.1 Exception Page Template

What is the exception template?

The "exception template" is the template file which is rendered once there is an unexpected exception being thrown by one of the payment gateways.

Why to change the exception template?

There might be a reason to change that template in case there is a need to display a custom message to the user or to change the overall template's appearance.

How to override exception template?

The default template location is src/PH/Bundle/PayumBundle/Resources/views/exception.html.twig.

To override the template, just copy the `exception.html.twig` template from the Payum bundle to `app/Resources/PHPayumBundle/views/exception.html.twig` (the `app/Resources/PHPayumBundle` directory won't exist, so you'll need to create it). You're now free to customize the template.

```
{# app/Resources/PHPayumBundle/views/exception.html.twig #}

<h1>My custom exception template.<h1>
```

That's it! A new template will be rendered whenever there will be an unexpected exception thrown.

3.6.2 Payment Thank Tou Template

What is the payment thank you template?

The "payment thank you template" is the template file which usually renders a "thank you" text to the user. A user is redirected to the "thank you URL" after the payment was completed or failed using one of the payment gateways. The content of that file is then rendered.

Why to change the payment thank you template?

There might be a reason to change that template in case there is a need to display a custom message to the user or to change the overall template's appearance.

How to override thank you template?

The default template location is src/PH/Bundle/CoreBundle/Resources/views/thankYou.html.twig.

To override the template, just copy the `thankYou.html.twig` template from the Core bundle to `app/Resources/PHCoreBundle/views/thankYou.html.twig` (the `app/Resources/PHCoreBundle` directory won't exist, so you'll need to create it). You're now free to customize the template.

```
{# app/Resources/PHCoreBundle/views/thankYou.html.twig #}

<h1>Thank you.<h1>
{% if app.request.get('token') %}
    <div style="color: #3D7700">Transaction id {{ app.request.get('token') }}</div>
{%  endif %}
```

The `token` query parameter is automatically appended to the "thank you URL", thus as you can see above you can read it in the "thank you template" to inform user about the transaction identifier.

That's it! A new template will be rendered whenever the payment process will be completed or failed.

Even if the the payment process failed, there still will be a redirect to "thank you" url.

Lear More

- *How To Change Thank You URL?*
- *How To Change Cancel URL?*

3.6.3 Payment Cancellation Template

What is the cancellation template?

The "cancellation template" is the template file which usually renders a payment's cancellation text to the user. A user is redirected to the "cancel URL" after the payment was cancelled using one of the payment gateways. The content of that file is then rendered.

Why to change the cancellation template?

There might be a reason to change that template in case there is a need to display a custom message to the user or to change the overall template's appearance.

How to override cancellation template?

The default template location is src/PH/Bundle/CoreBundle/Resources/views/cancel.html.twig.

To override the template, just copy the `cancel.html.twig` template from the Core bundle to `app/Resources/PHCoreBundle/views/cancel.html.twig` (the `app/Resources/PHCoreBundle` directory won't exist, so you'll need to create it). You're now free to customize the template.

```
{# app/Resources/PHCoreBundle/views/cancel.html.twig #}

<h1>The payment has been cancelled.<h1>
{% if app.request.get('token') %}
```

(continues on next page)

(continued from previous page)

```
    <div style="color: #3D7700">Transaction id {{ app.request.get('token') }}</div>
{%  endif %}
```

The `token` query parameter is automatically appended to the "cancel URL", thus as you can see above you can read it in the "cancel template" to inform user about the cancelled transaction identifier.

That's it! A new template will be rendered whenever the payment process will be cancelled.

Lear More

- *How To Change Thank You URL?*
- *How To Change Cancel URL?*

3.6.4 SEPA Direct Debit Bank Account Template

What is the SEPA Direct Debit Bank Account template?

The "SEPA Direct Debit Bank Account template" is the template file which is rendered when the SEPA Direct Debit payment method using Mollie gateway is used.

Once this payment method is selected/used the SEPA Direct Debit Bank Account template is rendered so the user can enter bank account details in order to process further payment.

Why to change the SEPA Direct Debit Bank Account template?

There might be a reason to change that template in case there is a need to display a custom message to the user or to change the overall template's appearance.

How to override SEPA Direct Debit Bank Account template?

The default template location is src/PH/Bundle/PayumBundle/Resources/views/Action/obtain_sepa_bank_account.html.twig.

To override the template, just copy the `obtain_sepa_bank_account.html.twig` template from the Payum bundle to `app/Resources/PHPayumBundle/views/Action/obtain_sepa_bank_account.html.twig` (the `app/Resources/PHPayumBundle` directory won't exist, so you'll need to create it). You're now free to customize the template.

Note: This documentation assumes you have already a working knowledge of the Symfony Forms. If you're not familiar with Symfony Forms, please start with reading the Forms and How to Customize Form Rendering from the Symfony documentation.

```
{# app/Resources/PHPayumBundle/views/Action/obtain_sepa_bank_account.html.twig #}

<form method="post" action="{{ actionUrl }}">
    {{ form_row(form) }}

    {{ form_rest(form) }}

    <input type="submit" value="Submit" />
</form>
```

As you can see above, the `actionUrl` variable is defined which should be used as a form action. This variable defines an action URL to which the submitted form data will be sent.

There is also a `form` variable which represents a Symfony Form object which can be customized according to the How to Customize Form Rendering from the Symfony documentation.

That's it! A new template will be rendered whenever the SEPA Direct Debit using Mollie gateway will be selected as a payment method.

3.6.5 SEPA Direct Debit Mandate Confirmation Template

What is the SEPA Direct Debit Mandate Confirmation template?

The SEPA Direct Debit Mandate Confirmation template is the template file which is rendered when the SEPA Direct Debit payment method using Mollie gateway is used.

Once this payment method is selected/used and the SEPA Direct Debit bank account details are submitted the SEPA Direct Debit Mandate Confirmation template is rendered where a user needs to confirm the SEPA Direct Debit Mandate in order to process further recurring payments.

Why to change the SEPA Direct Debit Mandate Confirmation template?

There might be a reason to change that template in case there is a need to display a custom message to the user or to change the overall template's appearance.

How to override SEPA Direct Debit Mandate Confirmation template?

The default template location is src/PH/Bundle/PayumBundle/Resources/views/Action/sepa_mandate_confirmation.html.twig.

To override the template, just copy the `sepa_mandate_confirmation.html.twig` template from the Payum bundle to `app/Resources/PHPayumBundle/views/Action/sepa_mandate_confirmation.html.twig` (the `app/Resources/PHPayumBundle` directory won't exist, so you'll need to create it). You're now free to customize the template.

```
{# app/Resources/PHPayumBundle/views/Action/sepa_mandate_confirmation.html.twig #}

<h3>SEPA Direct Debit Mandate</h3>

<p>By providing your IBAN and confirming this payment, you are authorizing "Your␣
↪Company Name" and Mollie,
    our payment service provider, to send instructions to your bank to debit your␣
↪account and your bank to debit
    your account in accordance with those instructions.
    You are entitled to a refund from your bank under the terms and conditions of␣
↪your agreement with your bank.
        A refund must be claimed within 8 weeks starting from the date on which your␣
↪account was debited.</p>

<h4>Creditor details:</h4>
<table border="1">
    <tr>
        <th>Creditor Identifier</th>
        <td>Your creditor id</td>
    </tr>
    <tr>
```

(continues on next page)

(continued from previous page)

```
        <th>Name</th>
        <td>Creditor name</td>
    </tr>
    <tr>
        <th>Address</th>
        <td>Creditor Address</td>
    </tr>
</table>

<h4>Customer details:</h4>
<table border="1">
    <tr>
        <th>IBAN</th>
        <td>{{ model.details.consumerAccount }}</td>
    </tr>
    <tr>
        <th>Holder name</th>
        <td>{{ model.details.consumerName }}</td>
    </tr>
</table>

<p>Date of signing: {{ model.signatureDate }}</p>


<form method="post" action="{{ actionUrl }}">
    <input type="hidden" name="mandate_id" value="{{ model.id }}"/>

    <button type="submit">Confirm</button>
</form>

<a href="{{ cancelUrl }}">Go Back</a>
```

As you can see above, the `actionUrl` variable is defined which should be used as a form action. This variable defines an action URL to which the submitted form data will be sent.

The `cancelUrl` variable prints the payment cancellation URL. If a user decides to go to that link, the payment process will be cancelled and a *Payment Cancellation Template* will be rendered.

There is also a `model` variable which is an array containing the details of the currently processed data from where you can get info about the currently submitted mandate.

Note: You can use a `dump` (e.g. `{{ dump(model) }}`) function to dump the value of `model` variable to find out what keys and values it contains. Note that `dump` function can be used only in the development environment.

That's it! A new template will be rendered whenever the SEPA Direct Debit using Mollie gateway will be selected as a payment method and when the bank account details form will be submitted.

- *Exception Page Template*
- *Payment Thank Tou Template*
- *Payment Cancellation Template*
- *SEPA Direct Debit Bank Account Template*
- *SEPA Direct Debit Mandate Confirmation Template*
- *Exception Page Template*

- *Payment Thank Tou Template*
- *Payment Cancellation Template*
- *SEPA Direct Debit Bank Account Template*
- *SEPA Direct Debit Mandate Confirmation Template*

Payum extensions

3.7 Payum Extensions

Payment workflows in Payments Hub are handled using Payum open-source library.

In order to make the use of the Mollie and Mbe4 payment gateways, we developed the following Payum extensions:

- payhelper/payum-mollie
- payhelper/payum-mbe4

API Documentation:

API Documentaton can be found at http://hub-docs.s-lab.sourcefabric.org.

How to Run Tests?

How to Translate the Payments Hub?

3.8 How to Run Tests?

Note: Running tests will only work in the development environment.

The behaviour of the application is covered using Behat scenarios.

Just run the following command:

```
bin/behat --strict --no-interaction -vvv -f progress
```

All Behat scenarios are located inside `features` directory inside Payments Hub repository.

3.9 How to Translate the Payments Hub?

Note: From now on, you can also help translate the Payments Hub using Crowdin platform https://crwd.in/payments-hub.

There are just a few translation files which contain the translations strings.

- messages.en.yml - contains generic payment translation strings
- PayumBundle.en.yml - contains translation strings related to the payment gateways
- validators.en.yml - contains translation strings related to the validation

To translate these files, for example, to German language, just copy and paste them into the same directory by changing the `en` to `de` in the file name (e.g. `messages.de.yml`, `PayumBundle.de.yml`), where `de` is the language code according to the ISO 639-1 and the ISO 3166-1 alpha-2 standards.

Then, just change the English values of the translation parameters to German.

Last but not least, set the `locale` parameter in `app/config/parameters.yml` file to `de` in order to load newly translated strings.

```
# app/config/parameters.yml
parameters:
    # ..
    locale: de
```

Note: You might need to clear the cache so the new files with the translations strings can be loaded properly. Just run command `php app/console cache:clear`.

Finally, commit these newly created files and open a Pull Request to the Payments Hub Repository.