PrestaShop module

Developers guide for Klarna Payments module

Module version 1.5.5
Technical guide version V 1.2.0
2024-03-27

1. File structure

The module's file structure has been organized into different folders, each containing specific files and services that support the module's functionality.

- The Controllers folder contains two subfolders: front and admin. The front folder contains all the module's front controllers, which handle requests coming from the module's front-end. On the other hand, the admin folder contains all the module's admin controllers, which handle requests coming from the back-office.
- The Src folder contains various PHP services that support different parts of the module's logic. These services are organized into different subfolders based on their functionality, such as Api, Core, Infrastructure, and Presentation. Each subfolder contains specific services that handle different parts of the module's functionality.
- The Views folder contains three main subfolders: css, js, and templates. The css folder contains the module's stylesheet files, which define the module's visual style. The js folder contains the module's front-end logic javascript files, which handle user interactions on the front-end. The templates folder contains the module's smarty template files, which define how the module's content is rendered on the front-end.
- The Vendor folder contains the module's external packages, which are defined in the composer.lock file. These external packages are used to support the module's functionality and have been added as dependencies to the module.
- Finally, the KlarnaPayment.php file is the module's main entry file that defines the core logic for the module. This file initializes the module's services and sets up the necessary configurations to make the module work correctly.

2. Main codebase entry points

Below we described three main entry paths to the module. Single core entry file and admin/front controllers.

2.1. KlarnaPayment class (klarnapayment.php)

Class is the main entry point for the module. This file is where PrestaShop enters the module's program. Some important methods inside this class include:

2.1.1. install() and uninstall()

In addition to the default parent::install() method, a separate service Installer is used in these methods. This service installs configuration values, database tables, hooks, and custom module order states. The uninstall method, on the other hand, uninstalls database tables and removes configuration values from the database. However, API credentials are left in the database.

2.1.2. getTabs()

This method returns back-office module tabs (admin controllers). The module is used by the default install() method to install these tabs. Without this method, admin controllers wouldn't be reachable.

2.1.3. getService()

This method uses a separate dependency injection library from the one used in PrestaShop.

2.1.4. getContent()

This method is the main entry point for back-office module configuration tabs.

2.1.5. hookPaymentOptions()

This method uses the PaymentOptionsPresenter service, in which, via an API call, available payment options are retrieved. If the region is not supported, no options will be returned.

2.1.6. hookDisplayAdminOrder()

This method displays the order actions block on the back-office single order page. The displayed block allows actions, such as capture payment, refund payment, cancel order, and such.

2.1.7. hookActionAdminControllerSetMedia()

This method registers the required CSS and JS assets for pages that require them. Additionally, the method calls PruneOldRecordsAction to remove module logs after x days pass from the last clean-up. Moreover, this method is responsible for displaying informational messages for back-office single order page actions (actions that are made in the module's custom actions block).

2.1.8. addFlash()

This method adds a notification message for various pages.

2.2. AdminKlarnaPaymentSettingsController (/controllers/admin/AdminKlarnaPaymentSettingsController.php)

Class is an important module controller that shows the main module settings, such as environment select, credentials input, order status mapping, and logging enable switch. The important methods used in this controller (same methods are used in every admin controller) are:

2.2.1. initContent()

This method displays content for the page. Inside the method, other class methods are called to display different page parts. Different methods are used to separate logic.

2.2.2. postProcess()

This method is called after each "save" button click. Inside the method, by checking for the submitted form name, various information is saved into the database.

2.3. AdminKlarnaPaymentOrderController (/controllers/admin/AdminKlarnaPaymentOrderController.php)

This is a hidden module’s admin controller responsible for back-office order page actions such as capture, refund, and cancel. It allows the administrator to manage Klarna payment orders and perform actions related to them.

2.4. AdminKlarnaPaymentStylingController (/controllers/admin/AdminKlarnaPaymentStylingController.php)

This class is responsible for showing the module's coloring settings in the "Styling" tab of the module's "configure" settings. The color scheme displayed in checkout to display payment options and payment modal can be set here.

2.5. AdminKlarnaPaymentLogsController (/controllers/admin/AdminKlarnaPaymentLogsController.php)

Class is a hidden module's admin controller responsible for displaying detailed event logs related to the Klarna payment module. It is accessed via the "Logs" tab in the module's configuration settings. The class contains two important methods:

2.5.1. __construct()

This method defines the table columns and how information from the database should be fetched via SQL queries.

2.5.2. displayAjaxGetLog()

This method is called via AJAX when a user clicks on the "context", "request", or "response" columns button to view more detailed information about a log entry. It fetches the detailed information from the database and returns it to the user's browser to be displayed.

2.6. KlarnaPaymentAuthorizationModuleFrontController (/controllers/front/authorization.php)

Class is called by Klarna as an endpoint on specific payment option cases when a customer is being redirected to another page. The class uses the CheckoutController service to create an order on the PrestaShop side.

2.7. KlarnaPaymentOrderModuleFrontController (/controllers/front/order.php)

Class is used for internal calls and is called via AJAX to retrieve order data on order submit action. The retrieved order data is sent to Klarna to initialize the payment process.

2.8. KlarnaPaymentPaymentModuleFrontController (/controllers/front/payment.php)

Class is also used for internal calls and is called on form submit as the form's action. This class uses the same CheckoutController service as authorization.php to create an order on the PrestaShop side. The only difference from authorization.php is that this class is called in the same runtime immediately after the Klarna payment modal closes and the customer is not being redirected from the PrestaShop checkout.

2.9. KlarnaPaymentSuccessModuleFrontController (/controllers/front/success.php)

Class is used for internal calls and finishes the order flow. If the request to the class had the correct security token, then the order-confirmation page will be shown after. Otherwise, the customer will be redirected to the order page.

3. Main module services (src)

3.1. Api

This folder contains several different services that are used for building API calls. These services are used to interact with external APIs and retrieve data, as well as to process and manipulate that data as needed within the module.

3.1.1. Apis

This folder contains a set of services that are specifically designed to make API calls and retrieve responses from external APIs. These services are responsible for handling the communication between the module and the external API, and for processing the response data as needed.

3.1.2. Requests

This folder contains a set of models that are used to transfer request information from the module to the external API. These models define the structure of the data being sent in the request, and ensure that the data is formatted correctly for the API to process.

3.1.3. Responses

This folder contains a set of models that are used to transfer response information from the external API back to the module, and ultimately to the client. These models define the structure of the data being returned by the API, and ensure that the data is formatted correctly for the module to process and use as needed.

3.2. Core

This folder is the heart of the module's functionality and contains various services used to handle different parts of the module's logic. It encompasses the majority of the business logic of the module and is responsible for processing data and handling the various interactions between different parts of the module. The services in this folder work together to create a cohesive whole and enable the module to function as intended.

3.2.1. Merchant

This folder contains various services that handle logic for merchant-related actions, such as credential manipulation. These services are responsible for handling the interactions between the module and the merchant's account, such as authentication and authorization. They provide a secure and reliable way for the module to communicate with the merchant's account and ensure that only authorized actions are taken.

3.2.2. Order

This folder contains various services that handle logic for manipulating created orders on the PrestaShop and Klarna sides. The services in this folder are responsible for creating, updating,

and managing orders, as well as handling any errors or issues that may arise. They ensure that the order creation process is seamless and error-free, and that all necessary data is collected and transmitted correctly.

3.2.2.1. Action

This folder contains services that handle specific single actions. These services are designed to handle a single specific action, such as creating an order or updating a customer's information. They are highly specialized and focused on a specific task, making them efficient and reliable.

3.2.2.2. Processor

This folder contains services that handle multiple logic blocks. The services in this folder are designed to handle complex and multi-step processes, such as creating a new order and updating customer information at the same time. They are responsible for coordinating the various services and processes involved in these tasks, ensuring that they are completed in the correct order and with the correct data.

3.2.3. Payment

This folder contains various services that handle logic for manipulating payments before the order is created on the PrestaShop and Klarna sides. These services are responsible for verifying payment information, calculating taxes and fees, and handling any other financial transactions involved in the order creation process. They ensure that all financial transactions are secure and accurate, and that the merchant and customer are charged and paid the correct amount.

** *The payment folder also contains the Extra Merchant Data (EMD) implementation see examples below.*

3.3. Infrastructure

This folder contains various services used to support the module's infrastructure. These services are responsible for handling the various tasks involved in setting up and maintaining the module, such as database management, configuration settings, and other low-level tasks. They are critical to the functioning of the module and ensure that it operates smoothly and efficiently.

3.3.1. Api

This folder contains services that handle API responses. These services are responsible for parsing and interpreting data received from API calls, as well as formatting and returning data to the API in the correct format. They ensure that the module can communicate effectively with external APIs and that data is transmitted correctly and reliably.

3.3.2. Bootstrap

This folder contains services that handle additional data installation and uninstallation for the module. These services are responsible for setting up and tearing down the various parts of the

module, such as database tables and configuration settings. They ensure that the module can be installed and uninstalled easily and without causing any issues or conflicts with other modules or software.

3.3.3. Container

This folder contains services that provide dependency injection services and providers for the module. These services are responsible for managing the various dependencies and services required by the module, such as database connections and API clients. They ensure that the module can access the necessary resources and services it needs to operate correctly, and that these resources are managed efficiently and securely.

3.4. Presentation

The Presentation folder contains various services that are responsible for rendering the front-end assets or visual blocks. These services are used to create an interface for the user to interact with the module. The folder may include services that render templates, forms, or other user interface elements. By separating the presentation logic from the core and infrastructure logic, it is easier to maintain and modify the user interface without affecting the underlying functionality.

** Custom Extra Merchant Data implementation

Currently the Klarna Payments module allows us to send extra merchant data (EMD) to Klarna for a better risk assessment. When the EMD setting is turned on, it triggers the request to automatically attach and send customer account info for Klarna according to their attachment schema (see the link below). However, by default, the module executes the "klarnapaymentAdditionalEmd" hook and forms the attachment from the returned array. We allow merchants/developers to implement and use the hook to form the attachment information according to their shop needs.

https://docs.klarna.com/api/attachment-schema/

https://docs.klarna.com/klarna-checkout/popular-use-cases/extra-merchant-data/#emd-packages/

Below we provide an example of how to implement the "klarnapaymentAdditionalEmd" hook:

```
public function hookKlarnapaymentAdditionalEmd(): array
{
      return [
            'subscription' => [
                  [
                        'subscription_name' => 'Contact_lenses',
                        'start_time' => '2020-11-24T15:00',
                        'end_time' => '2020-11-24T15:00',
                        'auto_renewal_of_subscription' => false,
                  ]
            ]
      ];
}
```

** Custom Order Line hook implementation

Currently the Klarna Payments module allows us to add additional order lines to the order request by registering the “klarnapaymentAdditionalOrderLineData” hook. The hook allows you to have custom surcharges and other fees or logic applied to the cart and still let your customers pay with Klarna. The only thing that you need to do is register the provided hook and provide information about the custom charge by filling the required fields in the required order_line format that you can check in the link below.

https://docs.klarna.com/api/payments/#operation/createOrder!path=order_lines&t=request

Here is an example of how the hook “klarnapaymentAdditionalOrderLineData” should be implemented (for amount use float values):

```
public function hookKlarnapaymentAdditionalOrderLineData(): array
{
      return [
            'name' => 'Cart tax',
            'quantity' => 1,
            'total_amount' => 2.50,
            'unit_price' => 2.50,
      ];
}
```

** Request customization hook implementation

Currently the Klarna Payments module allows us to alter the sent data for klarna by registering the “klarnaModifySessionRequest” and “klarnaModifyOrderRequest” hooks. The hooks allow you to customize the order details that are sent to Klarna according to your store needs and logic. The only thing that you need to do is register the provided hooks and modify the set data accordingly. More information about the fields and requests in the documentation links below.

*Note: you must register both of the given hooks and apply the same changes for repeating fields ex. (country, currency, locale etc.).

https://docs.klarna.com/api/payments/#operation/createCreditSession

https://docs.klarna.com/api/payments/#operation/createOrder

Here is an example of how the hooks “klarnaModifySessionRequest” and “klarnaModifyOrderRequest” should be implemented:

```
public function hookKlarnaModifySessionRequest($params)
{
    /** @var \KlarnaPayment\Module\Api\Models\Session $session */
    $session = $params['sessionRequest'];

    $session->setLocale('en-US');
    $session->setPurchaseCountry('US');
    $session->setPurchaseCurrency('USD');
}
```

You must match the “klarnaModifyOrderRequest” accordingly.

```
public function hookKlarnaModifyOrderRequest($params)
{
     /** @var \KlarnaPayment\Module\Api\Requests\CreateOrderRequest $request */
     $request = $params['orderRequest'];

     $request->setLocale('en-US');
     $request->setPurchaseCountry('US');
     $request->setPurchaseCurrency('USD');
}
```

*Warning: Modifying Klarna payment data requires a strong understanding of the Klarna API and potential consequences. Incorrect modifications can lead to payment processing errors or unexpected behavior. Proceed only if you are confident in your technical abilities.