SAP Commerce 1808 Connector R6.0 - Implementation Guide

- Reader's guide
- Implementation Guide
    - Introduction
        - Fundamental Concepts
        - Installation and Usage
        - Worldpay Accounts
        - Make the tests available in the storefront
        - Worldpay Magic Values
    - Notification Endpoint
        - Mocking Worldpay Notifications
    - Configuring the AddOn
        - Merchant Configuration
        - Checkout Flow
        - Alternative Payment Methods - B2C only
        - Client Side Encryption (CSE)
        - Using Merchant Tokens
        - Configuring the Support Email
        - Configuring the payment status inquiry retrievability
    - Extending the AddOn
        - Cron Jobs to Process Order Modifications from Worldpay
        - orderModificationProcessorJob
        - CronJobs to capture Payment Method Information
        - Riskscore and fraud
        - Notification commands
        - Cancel notification
        - Cancellations
        - Refunds
        - Checkout Flow
        - Customising the Worldpay AddOn
    - Project Customisation Points of Consideration
    - Supported Environments

Reader's guide

This implementation guide will cover the use of Worldpay as a payment provider in both a B2C and B2B context. Sections that only apply to either B2C or B2B will be marked as ***B2C only*** or ***B2B* only.**
The OCC AddOn supports a B2C scenario where payment details are supplied using Client Side Encryption (CSE, see below).

Implementation Guide

Introduction

Hybris is a java based eCommerce platform built on the Spring MVC framework. The purpose of providing an AddOn for Worldpay is to aid integration of the Worldpay payment gateway into a Hybris implementation. This document describes how to install and configure the AddOn to work with any Hybris implementation. This AddOn can either be a B2C or a B2B AddOn.

As Hybris is built on the Spring framework this makes it highly customisable and extensible. The plugin also utilises this framework so can also easily be extended to add specific behaviour if required.

Fundamental Concepts

Hosted Order Page (HOP) - *B2C only*

The plugin has been developed to support payment by redirecting to Worldpay Hosted Payment pages, also known as HOP.

The redirect payment model enables a site to reduce PCI considerations by ensuring card details are not visible to any aspect of their Hybris installation. Worldpay hosts the pages that capture and process the card details, all that is required of the implementer is to redirect the user to these pages at the relevant point in their checkout flow and then process the authorisation response returned from the hosted order pages.

Client Side Encryption (CSE)

Client-side encryption is a way of capturing the payment details of a customer and sending encrypted to the Hybris Commerce Suite to pass through Direct XML integration to Worldpay. The payment details are encrypted in the client browser using the Worldpay CSE encryption JavaScript library.

Card Tokenisation

Worldpay enables their clients to tokenise cards of customers so that the same payment method can be reused for subsequent orders.

Once a payment method (Card) has been tokenised, the payment details are saved into the customer's session cart.
If the customer chooses to save the payment details performing the checkout, the payment details are stored in the customer's profile.

If the customer at a later time chooses to remove the payment details, a delete token request is sent to Worldpay.
No token delete request is sent for the tokens associated with payment details that are only stored on the customer's cart and later orders.

This functionality is available on the Hosted order pages and Client-Side Encryption.

Alternative Payment Methods - *B2C only*

Alternative Payment Methods (APM's) allow for payment types other than Credit or Debit Cards to be used through the Worldpay payment gateway such as Paypal, iDeal or Alipay. All Alternative Payments are processed as a redirect payment regardless whether the payment is completed via the Worldpay's Hosted Order Page (HOP) which is the standard flow or a selected bank's website in the case of APMs which support bank transfers.

Installation and Usage

Installing the Plugin using the provided recipes

The AddOn provides 3 gradle recipes to be used with the Hybris installer.

1. wp_b2c_acc with fulfilment functionality for both accelerator storefront and OCC web service.
2. wp_b2c_acc_occ with fulfilment functionality for both accelerator storefront and OCC web service to work with Spartacus.
3. wp_b2c_acc_oms with OMS functionality for both accelerator storefront and OCC web service.
4. wp_b2b_acc with fulfilment functionality for only accelerator storefront

The recipes are based on the b2c_acc_plus, b2c_b2b_acc_oms (using only b2c), and b2b_acc_plus recipes provided by Hybris.

The recipes can be found under the installer folder in the extracted ZIP.

Under wp_b2c_acc, wp_b2c_acc_occ , wp_b2c_acc_oms, and wp_b2b_acc folders there is 1 file, build.gradle, which is the recipe. Note that this recipe may be extended and used for the necessities of the solution.

Follow these steps to use the recipes: in a Hybris installation:

1. Extract from the supplied ZIP file the folder Hybris to your ${Hybris_BIN_DIR}
2. Since the recipe generates the local.properties file with the properties defined in the recipe, optionally you can add your local.properties to the customconfig folder.

In order to install the AddOn using one of the recipes, run the following commands:

```
#This will create a solution from the accelerator templates, and
install the addons.
Hybris_HOME/installer$ ./install.sh -r [RECIPE_NAME] setup
```

```
#This will build and initialize the platform
Hybris_HOME/installer$ ./install.sh -r [RECIPE_NAME] initialize
#This will start a commerce suite instance
Hybris_HOME/installer$ ./install.sh -r [RECIPE_NAME] start
```

Assisted Services Module (ASM)

The AddOn supports payments through the storefront using the ASM.

Hosts file entries - *B2C only*

In order for the storefront return URLs to resolve correctly on return from HOP pages, the following must be added to your hosts file:

127.0.0.1 apparel-uk.local apparel-de.local electronics.local

If using your own site:

127.0.0.1 your-site.local

URLs

The backoffice is accessible at https://localhost:9002/backoffice

The storefront is directly extended, and can be accessed through the following URLs:

https://electronics.local:9002/yacceleratorstorefront/electronics/en

https://apparel-uk.local:9002/yacceleratorstorefront/en

https://apparel-de.local:9002/yacceleratorstorefront/de

https://powertools.local:9002/yacceleratorstorefront/en

(*B2C only)* In case the solution needs to support iframe, it is necessary that the URLs set up in the implementation are Top Level Domains. It means that a .local domain will not work and the Iframe integration will raise an error. To solve it, your need to be like:

https://electronics.myshop.com:9002/yacceleratorstorefront/electronics/en

https://apparel-uk.myshop.com:9002/yacceleratorstorefront/en

https://apparel-de.myshop.com:9002/yacceleratorstorefront/de

And your DNS or hosts file need to resolve those URL.

Worldpay Accounts

To integrate properly with Worldpay, you need to have an account set up to integrate with. For this, the following steps need to be completed with your Worldpay Implementation Manager:

- Worldpay account
- Creation of merchant codes
- URL needs to be provided for the Hybris Commerce Suite Order Notification Endpoint

Other configuration can be carried out through the Worldpay Merchant Interface:

- Setting up action notification emails from Worldpay (if required)
- Setting up MAC (Message Authentication Code) for each merchant (if required)
- Setting account contact details

While setting up the merchant configuration, please note the limit per transaction, usually 5000 GBP. If the solution needs a higher limit, please contact your Worldpay Partner Manager.

Ask your Worldpay Integration Support Manager to enable

- Dynamic Interaction
- APMs you need for your implementation.
- Parameters in the ResultURL for your installationId

Make the tests available in the storefront

To be able to run the tests stored in worldpayaddoncommons it's necessary to set the property value testclasses.addonname which is declared in the project.properties file of the same extension. The value should be set as the same of the storefront name where the addon is being installed.

e.g.: If your storefront name is samplestorefront, set the value as:

- *testclasses.addonname*=samplestorefront

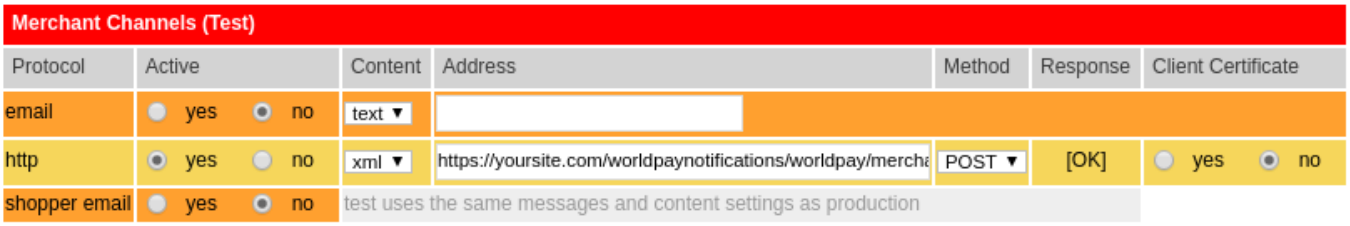
Worldpay Magic Values

In order to trigger different refused decline codes through the Worldpay sandbox, you can use magic values, which are specified on the link below:

http://support.worldpay.com/support/kb/gg/pdf/sandbox-magic-values.xls

Notification Endpoint

An endpoint to receive Worldpay notifications must be configured in the Worldpay merchant profile page:



The shipped configured endpoint is:

<public_FQDN>/worldpaynotifications/worldpay/merchant_callback

And it is set up in OrderModificationController in the com.worldpay.worldpaynotifications.controller.order.notification package inside the *worldpaynotifications* extension. As the code to receive order notification messages is separated in its own extension, it is possible to install this on just a subset of nodes in your Hybris Commerce Suite installation.

For security reasons, it is highly recommended to restrict access to the notification endpoint to only allow the Worldpay IP range. One of many solutions is to add a filter in the gateway, only whitelisting the Worldpay IP range accessing the nodes running the worldpaynotifications. A high-level architecture diagram can be found here: High Level Architecture

Mocking Worldpay Notifications

A mock to simulate notifications being sent from Worldpay has been produced and is included within the AddOn. This makes it possible to test various scenarios on environments that are not configured for getting notifications from Worldpay.

The mock can be accessed from the URL:

https://electronics.local:9002/worldpayresponsemock/responses

After placing an order, access the mock tool, and put in the Worldpay ordercode in the ordercode field. The Worldpay ordercode is the same as the id of the paymentTransaction attached to the order.

The responses configured in the mock will be sent to the endpoint, explained above.

The folder that contains the mock is *ext-worldpaytest* in the shipped zip file. If the mock is going to be used:

1. Add in your localextensions.xml <extension name="worldpayresponsemock"/>
2. Add in your local.properties -Djavax.xml.accessExternalDTD=all to the tomcat.generaloptions property
3. In Hybris/bin/platform, run "ant all"

> If the order has been submitted using the CSE/Direct flow, it is not necessary to mock the authorised response (will be ignored), as the response comes synchronously from Worldpay.

Configuring the AddOn

Merchant Configuration

There are three types of merchant codes which can be configured:

- Web (ECOM) - Used for purchases made via the Web channel using a Desktop computer
- Assisted Services Module / Customer Service (MOTO) - Used for purchases made via the Customer Services channel (Customer not present transaction)
- Replenishment (RECUR) - Used to capture scheduled replenishment orders via cronjobs (Customer not present transaction)

> MOTO stands for Mail Order/Telephone Order

Configuration Details

The configuration is defined in XML files in the "worldpayapi/resources/merchants" folder. In this folder, two files are shipped with the addon: merchants.xml and merchants-dev.xml. The merchants.xml file makes sure that all files in the folder matching the regex "merchants-*.xml" are imported, and the merchants-dev.xml contains a set of default merchants.

Once you have received details about your merchants, you will need to configure this in the XML files mentioned. This can be achieved in many ways, the simplest being editing the merchants-dev.xml file. For more advanced configurations, we recommend using spring profiles (see http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-definition-profiles-xml).

Switching MOCK/TEST/PRODUCTION environment:

In your local.properties set the value of the following property to change between environments. By default, in the *worldpayapi* extension it is configured to TEST

```
# Valid values for environment are MOCK, TEST and PRODUCTION
worldpay.config.environment=TEST
```

The actual endpoints are configured in the following properties:

```
worldpay.config.endpoint.MOCK=http://electronics.example.com:9001
/worldpayresponsemock/mock
worldpay.config.endpoint.TEST=https://secure-test.worldpay.com/jsp
/merchant/xml/paymentService.jsp
```

```
worldpay.config.endpoint.PROD=https://secure.worldpay.com/jsp/merchant
/xml/paymentService.jsp
```

You will need to modify the value of "worldpay.config.endpoint.MOCK" to match the environment you have set up to run the mock.

Channel to Merchant Mapping

The AddOn provides the functionality to select a set of merchants for a specific site. To achieve this, you need to define the various merchants in your merchants.xml

```
<util:map id="worldpayMerchantConfigurationSite1" value-type="com.worldpay.config.merchant.
WorldpayMerchantConfigData">

<entry key="web" value-ref="webMerchantConfigurationSite1"/>

<entry key="asm" value-ref="customerServiceMerchantConfigurationASM"/>

<entry key="replenishment" value-ref="replenishmentMerchantConfiguration"/>

</util:map>
```

```
<util:map id="worldpayMerchantConfigurationSite2" value-type="com.worldpay.config.merchant.
WorldpayMerchantConfigData">

<entry key="web" value-ref="webMerchantConfigurationSite2"/>

<entry key="asm" value-ref="customerServiceMerchantConfigurationASM"/>

<entry key"replenishment" value-ref="replenishmentMerchantConfiguration"/>

</util:map>
```

In local.properties, you can configure which sites use which merchants, using the UID of the site:

```
worldpaymerchantconfig.[UID of site]=worldpayMerchantConfigurationSite1

worldpaymerchantconfig.[UID of site]=worldpayMerchantConfigurationSite2
```

Merchant Details per Channel

```
<bean id="customerServiceMerchantConfiguration" class="com.worldpay.config.merchant.
WorldpayMerchantConfigData">

<property name="code" value="MERCHANT1MOTO"/>

<property name="password" value="password3"/>

<property name="macValidation" value="true"/>

<property name="macSecret" value="passwordMac3"/>

<property name="installationId" value="1043270"/>
```

```
<property name="statementNarrative" value="Statement Narrative Text MERCHANT1MOTO"/>

<property name="orderContent" value="Order Content MERCHANT1MOTO"/>

<property name="threeDSFlexJsonWebTokenSettings" ref="threeDS2JsonWebTokenSettings"/>

</bean>
```

Configuration properties

Name	Description
code	Merchant Code
password	Merchant password set up in the Worldpay merchant interface
macValidation	Defines if MAC validation should be used.
macSecret	Merchant MAC (Message Authentication Code) password set up in the Worldpay merchant interface
installationId	Identification of the installation
statementNarrative	Text that is displayed on the shopper's statement.
orderContent	HTML order content
includedPaymentTypes	Payment types accepted through this channel/merchant
excludedPaymentTypes	Payment types excluded from this channel/merchant

If CSE is used, an additional property needs to be added to the merchant configuration:

Name	Description
csePublicKey	Public key generated using the Worldpay Merchant Interface

Further explanation of these fields can be found in http://support.worldpay.com/support/kb/gg/pdf/rxml.pdf

Assisted services

If your application uses the assisted services module, you will need to configure a specific merchant using the key "asm" as seen in the merchants.xml example above. This merchant has to be compatible with the payment flow used by the storefront. Typically a different merchant is used as you would want to run the storefront with a different setup (such as disabling 3D secure) when placing orders as an agent.

3DS2/3DS-Flex Configuration

Wether your application has activated 3DS or 3DS2, on the merchants.xml you can find on every WorldpayMerchantConfigData the attribute threeDSFlexJsonWebTokenSettings which contains the 3DS-Flex configuration bean. Each attribute of the threeDSFlexJsonWebTokenSettings bean in merchants.xml is configured through a property:

```
<bean id="threeDS2JsonWebTokenSettings" class="com.worldpay.config.merchant.
ThreeDSFlexJsonWebTokenCredentials">
<property name="iss"
value="#{configurationService.configuration.getProperty('worldpay.config.3dsecureflex.jwt.iss')}"/>
<property name="jwtMacKey"
value="#{configurationService.configuration.getProperty('worldpay.config.3dsecureflex.jwt.jwtmackey')}"
/>
<property name="orgUnitId"
value="#{configurationService.configuration.getProperty('worldpay.config.3dsecureflex.jwt.orgunitid')}"
/>
<property name="alg"
value="#{configurationService.configuration.getProperty('worldpay.config.3dsecureflex.jwt.alg')}"/>
<property name="eventOriginDomain" value="#{configurationService.configuration.getProperty('worldpay.
config.domain.TEST')}"/>
<property name="challengeUrl"
value="#{configurationService.configuration.getProperty('worldpay.config.3dsecureflex.jwt.
challengeurl')}"/>
<property name="ddcUrl"
```

```
value="#{configurationService.configuration.getProperty('worldpay.config.3dsecureflex.jwt.ddcurl')}"/>
</bean>
```

The location of the sample properties values are under worldpayapi/project.properties. You can change the properties as you please with the values Worldpay provide to you for production environments due to the values provided are samples for testing purposes:

```
worldpay.config.3dsecureflex.jwt.iss
worldpay.config.3dsecureflex.jwt.jwtmackey
worldpay.config.3dsecureflex.jwt.orgunitid
worldpay.config.3dsecureflex.jwt.alg
worldpay.config.3dsecureflex.jwt.challengeurl
worldpay.config.3dsecureflex.jwt.ddcurl
worldpay.config.3dsecureflex.jwt.returnurl
worldpay.config.3dsecureflex.challengepreference
worldpay.config.3dsecureflex.secondauthorisation.submit.url
```

In case of doubt you can check the test values in the following Worldpay documentation page https://beta.developer.worldpay.com/docs/wpg/directintegration/3ds2#testing.

Checkout Flow

In agreement with the use of Worldpay's Hosted Payment Pages the following properties should be set in **local.properties** to hide the Accelerator's Checkout Flow options and default to HOP - ***B2C only.***

```
storefront.show.checkout.flows=false

site.pci.strategy=HOP
```

Also, the debug information on the HOP page (displaying Merchant Code and the callback URLs) can be enabled/disabled setting the following property - ***B2C only:***

```
hop.debug.mode=true|false
```

The AddOn modifies the **AccountOrderDetailsShippingComponent** and **OrderConfirmationShippingComponent** to display more detailed information about the payment details of the selected order from the order confirmation page and the customer's order history page. If they are not available, they are not displayed.

To install this modified component into the specific contentCatalog, run the following impex, changing <YOUR_CONTENT_CATALOG> to the appropiate contentCatalog name:

```
$contentCatalog=<YOUR_CONTENT_CATALOG>

$contentCV=catalogVersion(CatalogVersion.catalog(Catalog.id[default=$contentCatalog]),CatalogVersion.
version[default=Staged])[default=$contentCatalog:Staged]



# This update modifies the existing AccountOrderDetailsShippingComponent jspInclude component (with
payment details in order history) by setting a new page existing in the worldpayaddon

INSERT_UPDATE JspIncludeComponent;$contentCV[unique=true];uid[unique=true];name;page

;;AccountOrderDetailsShippingComponent;Account Order Details Shipping Info Component;/WEB-INF/views/addons/worldpayaddon
/responsive/pages/account/worldpayAccountOrderDetailShippingInfo.jsp ;;OrderConfirmationShippingComponent;Order Confirmation
Shipping Info Component;/WEB-INF/views/addons/worldpayaddon/responsive/pages/account/worldpayAccountOrderDetailShippingInfo.jsp
```

Alternative Payment Methods - *B2C only*

These are non-card payment methods which have to be configured in the system. The APM configuration is stored in the **WorldpayAPMConfiguration** item type.

APM Configuration Attributes - *B2C only*

Attribute	Type	Required	Description	Notes
code	String	**Yes**	This is the APM code which matches Worldpay APM codes (i.e. SOFORT-SSL)	See here for Worldpay payment method codes
name	String (localised)	**Yes**	This is the localised APM name. The property will be used on the order confirmation / history page as well as in the Customer Support Backoffice Perspective	
description	String (localised)	No	Brief APM description.	
autoCancelPendingTimeoutInMinutes	Integer	**Yes**	Timeout in minutes before the order is auto-cancelled. If Worldpay Authorise notification is not received within this time interval, the order will be cancelled.	This is likely to happen because the user has selected a delayed payment method which required further user action and the action has not been taken on time. Default value: 2880 minutes (2 days)
bank	Boolean	**Yes**	Indicates whether the apm supports bank transfer	To enable APMs that support bank transfer, at least one active **BankConfiguration** should be linked to the APM. Default value: False
countries	Set<CountryModel>	No	Set of countries for which the APM is available	The Country is related to the user's shipping address
currencies	Set<CurrencyModel>	No	Set of currencies for which the APM is available	
currencyRanges	Set<WorldpayCurrencyRangeModel>	No	Set of currency ranges for which the APM is available	
automaticRefunds	Boolean	**Yes**	The APM can be refunded automatically	These fields should be set according to the abilities of the specific APM. Please contact your Worldpay implementation manager for more details.
bankTransferRefunds	Boolean	**Yes**	The APM can be refunded via a bank transfer	

Bank Configuration Attributes - *B2C only*

Attribute	Type	Required	Description	Notes
code	String	Yes	This is the bank code which matches Worldpay Shopper Bank Code (i.e. SNS_REGIO)	See here for Worldpay shopper bank codes
apm	WorldpayAPMConfiguration	Yes	This is the apm the configuration relates to	
name	String (localised)	No	The name of the bank	
description	String (localised)	No	The description of the bank	
active	Boolean	Yes	Indicates whether this bank is active for apm	Default value: False

APM Availability Rules - *B2C only*

The configuration properties such as countries, currencies and currencyRanges define whether an APM can be used or not as a payment method.

There are three rules which all have to be true in order to have an APM available to use for payment.

Country Rule - *B2C only*

This rule inspects the country of the shipping address and evaluates the rules. The rules are described as follows:

Countries Set	Shipping Country	Result
Empty	N/A	TRUE
Not empty	Not in the Countries Set	FALSE
Not empty	In the Countries Set	TRUE

> If the project requirement is to use the Billing address country, the Country rule can be implemented to match this requirement.

Currency Rule - *B2C only*

This rule inspects the Cart's currency and evaluates the rule. The rules are described as follows:

Currencies Set	Cart's Currency	Result
Empty	N/A	TRUE
Not empty	Not in the Currencies Set	FALSE
Not empty	In the Currencies Set	TRUE

Currency Ranges Rule - *B2C only*

This rule inspects the Cart's total and evaluates the rule. The rules are described as follows:

Currency Range Set	Cart's Currency	Currency Range Boundaries	Cart's Total	Result
Empty	N/A	N/A	N/A	TRUE
Not empty	Not in the Currency Range Set	N/A	N/A	TRUE
Not empty	In the Currency Range Set	No Min, No Max	N/A	TRUE
Not empty	In the Currency Range Set	Min and Max	< Min OR >Max	FALSE
Not empty	In the Currency Range Set	Min and Max	>= Min AND <=Max	TRUE

Currency Ranges are stored in the **WorldpayCurrencyRange** item type which has the following attributes:

Attribute	Type	Required	Description
currency	CurrencyModel	**Yes**	This is the Currency for the defined range
min	Double	No	This is the minimum amount for the defined range
max	Double	No	This is the maximum amount for the defined range

Bank Transfer Rule - *B2C only*

This rule inspects the APM's bank configuration and evaluates the rule. The rules are described as follows:

APM's Bank Indicator	Bank Configuration Set	Result
False	Not checked	TRUE
True	Does not contain active banks	FALSE

True	Contains active banks	TRUE

Bank information

The bank the customer chose in the transaction is saved against the PaymentTransaction in the property *worldpayBank*. This information helps customer support agents help customers with more information about the payment method they chose and troubleshoot possible issues.

Modifying APM Availability Rules - *B2C only*

Rules are executed by the APM Availability Service which is defined in **worldpayapi-spring.xml** as follows:

worldpayapi-spring.xml

```
<alias name="defaultAPMAvailabilityService" alias="
apmAvailabilityService" />
<bean id="defaultAPMAvailabilityService" class="com.worldpay.service.
apm.impl.DefaultAPMAvailabilityService">
        <property name="apmAvailabilityStrategyList">
            <list>
                <ref bean="apmAvailabilityCountryStrategy"/>
            <ref bean="apmAvailabilityCurrencyStrategy"/>
            <ref bean="apmAvailabilityRangeStrategy"/>
                        <ref bean="apmAvailabilityBankStrategy"/>
        </list>
    </property>
</bean>
```

The service runs the strategies defined in the **apmAvailabilityStrategyList** and exits as soon as a strategy returns a false result. The service is extensible and strategies can be added, removed or modified.

All strategies implement the **APMAvailabilityStrategy** interface and receive the **WorldpayAPMConfigurationModel** and the **CartModel** as input .

The following class diagram describes the implementation for the APM availability rules.

Alternative Payment Methods CMS implementation - *B2C only*

All templates, pages, components, etc. belong to a specific content catalogue.

The page templates for the billing address pages listed below contain the PaymentButtons content slot.

- WorldpayPaymentAndBillingCheckoutPageTemplate
- WorldpayCSEPaymentAndBillingCheckoutPageTemplate
- WorldpayIframePaymentAndBillingCheckoutPageTemplate

The following components for PaymentButtons are allowed: WorldpayCCComponent and the WorldpayAPMComponent.

WorldpayCCComponent extends the SimpleCMSComponent and only contains 1 extra attribute: a localised image used to represent the pay-by-card option.

WorldpayAPMComponent also extends the SimpleCMSComponent but contains 2 additional attributes: a localised image to represent the APM and the configuration for the APM.

worldpayaddon-items.xml

```
<itemtype code="WorldpayCCComponent" autocreate="true" generate="true"
          extends="SimpleCMSComponent" jaloclass="com.worldpay.jalo.
WorldpayCCComponent">
    <attributes>
        <attribute type="localized:Media" qualifier="media">
            <persistence type="property" />
        </attribute>
    </attributes>
</itemtype>
<itemtype code="WorldpayAPMComponent" autocreate="true" generate="true"
          extends="SimpleCMSComponent" jaloclass="com.worldpay.jalo.
WorldpayAPMComponent">
<attributes>
    <attribute qualifier="apmConfiguration" type="
WorldpayAPMConfiguration">
        <persistence type="property"/>
    </attribute>
    <attribute type="localized:Media" qualifier="media">
        <persistence type="property" />
    </attribute>
</attributes>
```

The WorldpayCCComponent uses a Hybris renderer to populate the model (by default this will automatically populate the JSP with the Component's attributes). This is set up via spring:

worldpayaddon-web-spring.xml

```
<bean id="WorldpayCCComponentRenderer" parent="
addOnJspIncludeCMSComponentRenderer"/>
<bean id="WorldpayCCComponentRendererMapping" parent="
addonCmsComponentRendererMapping">
    <property name="typeCode" value="WorldpayCCComponent"/>
    <property name="renderer" ref="WorldpayCCComponentRenderer"/>
</bean>
```

The WorldpayAPMComponent uses a custom controller which extends the GenericCMSAddOnComponentController - which allows the Component's attributes to be available on the jsp (similar to how the WorldpayCCComponent works using the renderer). This is configured using Spring annotations:

WorldpayAPMComponentController

```
@Controller("WorldpayAPMComponentController")
@RequestMapping(value = "/view/WorldpayAPMComponentController")
public class WorldpayAPMComponentController extends
GenericCMSAddOnComponentController {...}
```

For the front-end to show the payment button components we have added the following to the worldpayChoosePaymentDetailsPage.jsp:

worldPayChoosePaymentDetailsPage.jsp

```
<cms:pageSlot position="PaymentButtons" var="button" element="div"
class="cms-payment-button">
    <cms:component component="${button}"/>
</cms:pageSlot>
```

The components themselves are represented by the 2 following JSPs:

worldpayapmcomponent.jsp
worldpaycccomponent.jsp

These contain simple radio buttons that are added in the PaymentDetailsForm along with a localised media.

At this point we have only checked to see if the components should display the apm on the front end, for backend validation we use the: PaymentDetailsFormValidator to do the same check as the GenericCMSAddOnComponentController (same rules as above):

PaymentDetailsFormValidator

```
@Component("paymentDetailsFormValidator")
public class PaymentDetailsFormValidator implements Validator {
...
   if (!errors.hasErrors() && !CREDIT_CARD_PAYMENT_METHOD.equals(form.
getPaymentMethod())) {
      final WorldpayAPMConfigurationModel apmConfiguration =
apmConfigurationLookupService.getAPMConfigurationForCode(form.
getPaymentMethod());
      if (!apmAvailabilityService.isAvailable(apmConfiguration,
cartService.getSessionCart())) {
         errors.rejectValue(FIELD_PAYMENT_METHOD, "worldpay.
paymentMethod.notAvailable", "Payment method is not available");
      }
   }
...
}
```

If CMS content configuration from worldpaysampledataaddon is used the WorldpayAPMComponent are restricted from being used in assisted service mode (ASM). This is done using the AssistedServiceSessionReversedRestriction defined in the assistedservicestorefront AddOn.

Apple Pay APM

The AddOn includes an Alternative Payment Method for Apple Pay. This payment method will be available in the checkout, as part of the list of APM's on Safari on Mac or on iPhone with Apple Pay (iPhone 6 or later). The Mac needs to configured to use Apple Pay, have a finger print scanner or should have a paired iPhone with Apple Pay close by. We've implemented version 5 of the Apple Pay JS API.

Activating Apple Pay Profile to enable this functionality

In order to enable Apple Pay you will need to add the *applepay* profile to your list of active Spring Profiles:

```
spring.profiles.active=foo,bar,applepay
```

Setting up Apple Pay for your webshop

- - Request Apple Pay integration from Worldpay. You will receive a CSR file from Worldpay after activation.
  - Create an account on https://developer.apple.com
  - Create a Merchant ID https://developer.apple.com/account/ios/identifier/merchant/create
  - Click Edit on the Merchant ID
    - Create a Payment processing certificate, you will need the CSR file you have received from Worldpay
    - Add your fully qualified domains.
      - Download the merchant verification file and host this file on your domain (https://your.domain.io/.well-known/apple-developer-merchantid-domain-association.txt)
      - Verify the domain in the Apple Developer Console.
    - Create a Merchant Identity Certificate in the Apple Developer Console. This will produce the **certFromApple.cer** file below.

Creating your applePaytls.key

```
openssl req -sha256 -nodes -newkey rsa:2048 -keyout applepaytls.key -
out applepaytls.csr
openssl x509 -inform der -in certFromApple.cer -out
merchant_identity_cert.pem
```

- Create your keystore

Creating p12 keystore

```
openssl pkcs12 -export -in merchant_identity_cert.pem -inkey
applepaytls.key -out apple-pay.p12 -name "Worldpay <your name> Apple
Pay keystore"
```

- Add the `applePayComponent` APMComponent to the `WorldpayPaymentButtonsSlot`

Setting up your server

- First, follow the instructions from Apple https://developer.apple.com/documentation/apple_pay_on_the_web/setting_up_your_server.
- Configure the merchant configuration in beans.xml.

Merchant configuration

```
    <bean id="applePaySettings" class="com.worldpay.config.merchant.
ApplePayConfigData">
        <property name="merchantId" value="xxxx" />
        <property name="merchantName" value="Worldpay test" />
        <property name="countryCode" value="PK" />
        <property name="merchantCapabilities">
            <util:list value-type="java.lang.String">
                <value>supportsCredit</value>
```

```
                <value>supportsDebit</value>
                <value>supports3DS</value>
                <value>supportsEMV</value>
            </util:list>
        </property>
        <property name="supportedNetworks">
            <util:list value-type="java.lang.String">
                <value>amex</value>
                <value>discover</value>
                <value>jcb</value>
                <value>maestro</value>
                <value>masterCard</value>
                <value>visa</value>
            </util:list>
        </property>
    </bean>
```

Property	Description
merchantId	The Worldpay merchant identifier
merchantName	The name of your webshop. This string is used in the Apple Pay payment sheet as the receiver of the money
countryCode	The two-letter ISO 3166 country code
merchantCapabilities	The supported values for merchantCapabilities are: • `supports3DS` - Required. This value must be supplied. • `supportsCredit` - Optional. If present, only transactions that are categorized as credit cards are allowed. • `supportsDebit` - Optional. If present, only transactions that are categorized as debit cards are allowed. • `supportsEMV` - Include this value only if you support China Union Pay transactions. https://developer.apple.com/documentation/apple_pay_on_the_web/applepaypaymentrequest/1916123-merchantcapabilities
supportedNetworks	List of all supported card networks https://developer.apple.com/documentation/apple_pay_on_the_web/applepayrequest/2951831-supportednetworks

- At the beginning of every payment request, the browser will require to validate the merchant. The server has to send an encrypted message to the Apple merchant validation services. You will have to use the `apple-pay.p12` file you have created during server setup. You have to configure the `worldpayApplePayHttpClient` to use your key store.

HttpClient configuration

```
<bean id="worldpayApplePayHttpClient" class="com.worldpay.web.client.
WorldPayApplePayHttpClientFactoryBean">
     <property name="password" value="changeit" />
     <property name="keyStoreType" value="PKCS12" />
     <!-- You will need to generate your own certificate -->
     <property name="certificateFile" value="classpath:applepaycert
/dummy-certificate.p12"/>
</bean>
```

Apple has a sandbox, with predefined payment cards that can be used to test with. You have to create a new Apple account especially for the sandbox, you can't register an email address to both the sandbox and 'normal' Apple services. You can invite users to be part of the sandbox in the Apple management console.

Adding a sandbox card can be complicated. You can only add cards using the Wallet app on the iPhone, log out of your account on your Mac before you do that to prevent errors. Make sure you have a billing address configured. Our experience is that using an address in the United States works, other countries might give problems. These problems are only related to the sandbox environment.

If you are testing with a Mac without fingerprint reader, you have to be logged into the same iCloud account on the Mac and the iPhone and the two have to be paired via bluetooth.

Google Pay APM

The AddOn includes an Alternative Payment Method for Google Pay. This payment method will be available in the checkout, as part of the list of APM's. The connector allows creating tokens for this payment method.

Setting up Google Pay for your webshop

- Add the `applePayComponent` APMComponent to the `WorldpayPaymentButtonsSlot`
- Request google pay to be enabled on production: https://services.google.com/fb/forms/googlepayAPIenable/

Setting up your server

Configure the merchant configuration in beans.xml.

Merchant configuration

```
    <bean id="googlePaySettings" class="com.worldpay.config.merchant.
GooglePayConfigData">
        <property name="cardType" value="CARD"/>
        <property name="allowedAuthMethods">
            <util:list value-type="java.lang.String">
                <value>PAN_ONLY</value>
                <!--<value>CRYPTOGRAM_3DS</value>-->
            </util:list>
        </property>
        <property name="allowedCardNetworks">
            <util:list value-type="java.lang.String">
                <value>AMEX</value>
                <value>DISCOVER</value>
                <value>JCB</value>
                <value>MASTERCARD</value>
                <value>VISA</value>
            </util:list>
        </property>
        <property name="environment" value="TEST"/>
        <property name="gatewayMerchantId" value="xxx"/>
        <property name="merchantId" value=""/>
        <property name="merchantName" value="#
{websiteMerchantConfiguration.code}"/>
    </bean>
```

Property	Description
cardType	CARD is the only supported value at this time https://developers.google.com/pay/api/web/reference/object#PaymentMethod
allowedAuthMethods	https://developers.google.com/pay/api/web/reference/object#CardParameters
allowedCardNetworks	https://developers.google.com/pay/api/web/reference/object#CardParameters
environment	The supported values for environment are: • `TEST`: the payment card is not charged • `PRODUCTION`
gatewayMerchantId	This value you will receive from Worldpay
merchantId	This property is only required for production, leave empty during test https://developers.google.com/pay/api/web/reference/object#MerchantInfo

Testing

You need enable Google Pay on your Google account. During test, you have to work use valid a credit/debit card. All payments will not be charged to your credit card.

Client Side Encryption (CSE)

The AddOn can be configured to use CSE for submitting orders to Worldpay. For this, a new Page Template has been added using a different front-end template.

template/cms-content.impex

```
INSERT_UPDATE PageTemplate;$contentCV[unique=true];uid[unique=true];
name;restrictedPageTypes(code);velocityTemplate[translator=de.Hybris.
platform.commerceservices.impex.impl.FileLoaderValueTranslator];active
[default=false];frontendTemplateName
;;WorldpayCSEPaymentAndBillingCheckoutPageTemplate;Worldpay CSE Payment
Page Template;WorldpayPaymentPage;$jarResourceCmsCockpit/structure-view
/structure_WorldpayCSEPaymentAndBillingCheckoutPageTemplate.vm;;addon:
/$addonExtensionName/pages/checkout/multi
/worldpayChooseCSEPaymentDetailsPage
```

CSE is enabled by changing the page template of the payment page as described in the end user guide. APM's will still use the redirect payment flow if CSE is enabled.

As CSE uses the Worldpay Direct XML integration method, the AUTHORISE is handled synchronously by Worldpay. The merchant interface can be configured to still send asynchronous AUTHORISE notifications, but these are not required for normal operation. The rest of the notifications work in the same way as in the redirect flow.

Using Merchant Tokens

Merchant tokens can be created either through the eCommerce channel or through the POS channel. The plugin supports the use and creation of merchant tokens. Its behaviour can be configured by site setting the property to true or false where applicable.

```
# site specific
worldpay.merchant.token.enabled.your-site

# globally accross all sites
worldpay.merchant.token.enabled
```

General configuration

The AddOn by default changes the **paymentProvider** of the store to Worldpay and associates the store with a specific **checkout group**.

These properties are changed with below Impex template when the AddOn is installed and the system is updated (with core data for your store) or initialised:

stores/template/store.impex

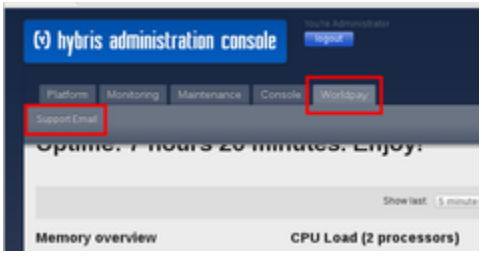
```
$paymentProvider=Worldpay
$checkoutGroup=worldpayCheckoutGroup

# Base Store
UPDATE BaseStore;uid[unique=true];paymentProvider;checkoutFlowGroup
;$storeUid;$paymentProvider;$checkoutGroup
```

Extension properties

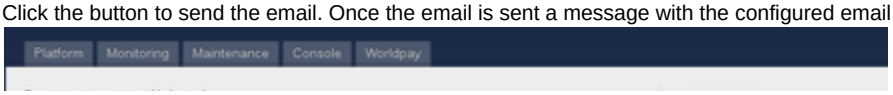
Extension	Properties	Description
worldpayresponsemock	Modification Controller Mapping	**worldpayresponsemock.order.notification.endpoint** should be changed in your environment specific local.properties if you want to use the mock. It has to match the mapping of the **com.worldpay.worldpaynotifications.controllers.order.notification.OrderModificationController**

Configuring the Support Email

There is a new tab in the HAC that sends the current configuration of the AddOn to Worldpay.



When selecting the 'Support Email' subtab a preview of the generated support email body will be displayed, along with a 'Send Email' button. Click the button to send the email. Once the email is sent a message with the configured email receiver is displayed.



The following properties should be configured in order to use this functionality:

Property	Description	Example
worldpay.support.email.address	The address the mail will be sent to.	plugins@worldpay.com
worldpay.support.email.display.name	Display name for the email client	Worldpay Plugin Support
worldpay.support.email.subject	Subject for the email	Worldpay Support for [StoreFrontName - CompanyName]
customer.support.email.address	Sender's email address	shop123@mail.com
customer.support.email.display.name	Sender's display name	Awesome Shop123
customer.support.email.address.reply.to	Email address to send the reply to	plugin-support@system.com

> For this functionality to work, it is expected that the SMTP server is configured.

The support email service uses a list of appenders to build the body of the email. To add more information to the support email, create an appender which implements **com.worldpay.support.appender.WorldpaySupportEmailAppender** and add it to the list of appender.

Below is the configuration for the email service used by the AddOn.

worldpayapi-spring.xml

```
<alias name="defaultWorldpaySupportEmailService" alias="
worldpaySupportEmailService"/>
<bean id="defaultWorldpaySupportEmailService" class="com.worldpay.
support.impl.DefaultWorldpaySupportEmailService">
    <property name="emailAppenders">
        <list>
            <ref bean="HybrisVersionAppender"/>
            <ref bean="currentTimeAppender"/>
            <ref bean="userDisplayNameAppender"/>
            <ref bean="merchantConfigurationAppender"/>
            <ref bean="configuredFlowsAppender"/>
            <ref bean="paymentTransactionAppender"/>
            <ref bean="extensionListAppender"/>
            <ref bean="clusterInformationAppender"/>
        </list>
    </property>
</bean>
```

To extend the list of appenders, use **listMergeDirective** as shown below.

Example of List Merge Directive

```
<bean id="myCustomAppenderListMergeDirective" depends-on="
worldpaySupportEmailService" parent="listMergeDirective">
    <property name="add" ref="myCustomAppender"/>
    <property name="listPropertyDescriptor" value="emailAppenders" />
</bean>
```

Configuring the payment status inquiry retrievability

Due to the possibility to configure merchants to not respond with the payment status as parameters on the resultURL after a transaction in HOP, the plugin has implemented a convenient fallback solution in case of not receiving it as part as the transaction response.

In the scenario where payment status is not in the gateway response and the gateway is not able to answer the payment inquiry immediately an order status inquiry retry will be triggered to address this situation. The plugin will inquiry the order status several times until the gateway is able to answer it or a timeout is produced, the number of retries and the delay between them can be configured.

The following properties can be configured in *worldpayapi* extension to support the order inquiry retrievability functionality:

Property	Description	Example
`worldpayapi.inquiry.max.number.of.retries`	Maximum number of tries to inquiry the order status	3
`worldpayapi.inquiry.delay.between.retries`	Delay between retries in seconds	3

Extending the AddOn

Cron Jobs to Process Order Modifications from Worldpay

Once an order modification is captured in the Hybris Commerce Suite, a series of Cron Job's have been created to process the messages. These include mission-critical jobs to capture responses to move orders through the fulfilment process as well as various monitoring and housekeeping jobs to notify Agents and Support of stuck orders or clean up processed data.

orderModificationProcessorJob

Each order modification from Worldpay is stored in the database. It is recommended that the job is run every minute for it to process all the pending modifications that the system has received. Once a modification has been processed, the corresponding order is looked up and an event is sent to launch the associated order process. Then the order modification is marked as processed. In case an error has occurred while processing the order modification, the modification is marked as defective.

Configuration

Property	Description
`typeOfPaymen`	**This is a Set of de.Hybris.platform.payment.enums.PaymentTransactionType entries. Currently, AUTHORIZATION, CAPTURE and CANCEL are supported by the plugin in the fulfilment flavour and the OMS flavour also supports SETTLED and REFUNDED. The cron job will only process the notification types found in this Set.**

`tTransactionToProcessSet`	**This will allows developers to create fine-grained cron jobs to target specific transaction types if they wish so. For example, there could be a dedicated cron job which processes AUTHORIZATION messages only (which are usually more frequent) having its own scheduled interval, and a second cron job to process all other messages.**

Impex

projectdataOrderModificationCronjob.impex

```
UPDATE GenericItem[processor=de.Hybris.platform.commerceservices.impex.
impl.ConfigPropertyImportProcessor];pk[unique=true]

$activateTriggers=$config-worldpayOrderSync.notification.received.
trigger.activate

INSERT_UPDATE OrderModificationCronJob;code[unique=true];job(code);
sessionLanguage(isoCode)[default=en];
typeOfPaymentTransactionToProcessSet(code)
;orderModificationProcessorJob;
orderModificationProcessorJobPerformable;;CAPTURE,AUTHORIZATION,CANCEL

INSERT_UPDATE Trigger;cronJob(code)[unique=true];second;minute;hour;day;
month;year;relative;active[default=$activateTriggers];maxAcceptableDelay
;orderModificationProcessorJob;0;1;-1;-1;-1;-1;true;;-1
```

cleanUpProcessedOrderModificationsCronJob

This job is in charge of cleaning up old order modifications that have been already successfully processed. By default the job runs daily and cleans up processed order modifications older than 5 days:

Configuration

Property	Description
`daysToWaitBeforeDeletion`	Indicates how old (in days) the Order Modification message has to be before it's cleaned up (deleted) from the system

Impex

The job and trigger are created with the following Impex script:

projectdataOrderCleanUpCronjob.impex

```
UPDATE GenericItem[processor=de.Hybris.platform.commerceservices.impex.
impl.ConfigPropertyImportProcessor];pk[unique=true]
```

```
$activateTriggers=$config-worldpayOrderSync.notification.cleanup.
triggers.activate

INSERT_UPDATE CleanUpProcessedOrderModificationsCronJob;code
[unique=true];job(code);sessionLanguage(isoCode)[default=en];
daysToWaitBeforeDeletion
;cleanUpProcessedOrderModificationsCronJob;
cleanUpProcessedOrderModificationsJobPerformable;;5

INSERT_UPDATE Trigger;cronJob(code)[unique=true];cronExpression;
relative;active[default=$activateTriggers];maxAcceptableDelay
;cleanUpProcessedOrderModificationsCronJob;0 0 0 * * ?;true;;-1
```

notifyUnprocessedOrderModificationsCronJob

If order modifications records haven't been processed after a certain time (configured maximum days limit - by default 7), a ticket per modification is created and assigned to the customer support agents for manual review. By default, the job runs daily.

Configuration

Property	Description
`unprocessedTimeInDays`	Indicates how old (in days) the Order Modification message can be unprocessed before a ticket is created. The message is not removed from the system, facilitating the support resources to inspect it and understand why the message has not been processed.

Impex

projectdataOrderNotificationCronjob.impex

```
UPDATE GenericItem[processor=de.Hybris.platform.commerceservices.impex.
impl.ConfigPropertyImportProcessor];pk[unique=true]

$activateTriggers=$config-worldpayOrderSync.notification.unprocessed.
trigger.activate

INSERT_UPDATE NotifyUnprocessedOrderModificationsCronJob;code
[unique=true];job(code);sessionLanguage(isoCode)[default=en];
unprocessedTimeInDays
;notifyUnprocessedOrderModificationsCronJob;
orderModificationUnprocessedModificationsNotifierJobPerformable;;7

INSERT_UPDATE Trigger;cronJob(code)[unique=true];cronExpression;
relative;active[default=$activateTriggers];maxAcceptableDelay
;notifyUnprocessedOrderModificationsCronJob;0 0 0 * * ?;true;;-1
```

CronJobs to capture Payment Method Information

In any XML Redirect integration, a job must be run to capture the details of the payment method that was used by the customer.

apmOrderTimeoutCronJob - *B2C only*

This job processes all the orders in PAYMENT_PENDING status that reached their timeout date set by paymentInfoTimeoutPreparationCronJob. It sets Payment Transaction Entries of those orders in the REVIEW status and wakes up the process. The new payment transaction entries status causes the Authorisation Failed Notification being sent to the customer and moving the process into the FAILED state.

Impex

projectdataAPMOrderTimeoutCronjob.impex

```
UPDATE GenericItem[processor=de.Hybris.platform.commerceservices.impex.
impl.ConfigPropertyImportProcessor];pk[unique=true]

$activateTriggers=$config-worldpayAPMOrder.timeout.triggers.activate

INSERT_UPDATE CronJob;code[unique=true];job(code);sessionLanguage
(isocode)[default=en]
;apmOrderTimeoutCronJob;apmOrderTimeoutJobPerformable;

INSERT_UPDATE Trigger;cronJob(code)[unique=true];cronExpression;
relative;active[default=$activateTriggers];maxAcceptableDelay
;apmOrderTimeoutCronJob;0 0/15 * * * ?;true;;-1
```

paymentInfoInquiryCronJob

This Cron Job captures details of the actual payment method used for the order (Card or APM) as well as triggers operational workflow parameters for APM's.

Since with many APM's a series of actions need to be completed for payment to be fulfilled, if these actions don't happen an auto-cancellation feature is necessary to ensure a properly completed order fulfilment workflow. Each APM can provide an auto cancellation timeout value (which is stored in minutes) for orders that haven't been processed. This job uses the timeout configured against the APM to set a timeout date for each order found in the system in **PAYMENT_PENDING** status. If no configured timeout is found for a specific APM, no timeout date will be set and therefore a default timeout (configured to 2 weeks) is used.

The job works in two steps. First, it retrieves payment transactions of pending orders without timeout date. Next, for each payment transaction, an Order Inquiry is sent to Worldpay to obtain the payment type of the payment associated with the transaction. The Payment type is saved and payment is converted accordingly - either to a **WorldpayAPMPaymentInfo,** if it is an APM, or to a **CreditCardPaymentInfo.** During the creation of the **WorldpayAPMPaymentInfo,** the timeoutDate is set if the timeout value is found in the corresponding APM configuration. The timeout date is calculated from the creation date of the payment transaction plus the timeout value configured for the APM.

> Please contact Worldpay to obtain the recommended timeout for each APM

The job also handles paymentTransactions that haven't received any notification from Worldpay within a given timeframe. The paymentTransactionEntries related to those paymentTransactions will be marked as REJECTED, and the reference fulfilment process will be re-triggered to cancel the customer's order.

Configuration

Property	Description

worldpay.apm.minutes.before.inquiring.timeout	The number of minutes for an APM payment transaction to wait for a notification from Worldpay before inquiring about auto cancel timeout.
worldpay.APM.days.before.stop.inquiring.timeout	The number of days before rejecting paymentTransactionEntries that have not received a notification from Worldpay and stop inquiring about the transaction.

Both properties take their default value from spring-configuration (below), but the spring-values can be overruled by the values set in **local.properties**.

worldpaynotifications-spring.xml

```
<bean id="paymentInfoInquiryJobPerformable" class="com.worldpay.cronjob.
PaymentInfoInquiryJobPerformable" parent="abstractJobPerformable">
        <property name="orderInquiryService" ref="orderInquiryService"
/>
        <property name="worldpayPaymentTransactionDao" ref="
paymentTransactionDao"/>
        <property name="worldpayMerchantService" ref="
worldpayMerchantService"/>
        <property name="configurationService" ref="configurationService"
/>
        <property name="paymentTransactionRejectionStrategy" ref="
paymentTransactionRejectionStrategy"/>
        <!-- Custom configuration -->
        <property name="defaultBlanketTimeInDays" value="5"/> <property
name="defaultWaitInMinutes" value="15"/>
</bean>
```

If the defaultBlanketTimeInDays is smaller (in time) than the defaultWaitInMinutes, the blanket timeout rule won't take effect until the def aultWaitInMinutes has passed. For instance, if the defaultBlanketTimeInDays is one day, and the defaultWaitInMinutes is 2880 (2 days), no payment transactions will be rejected before they're two days old.

Impex

projectdataPaymentInfoInquiryCronjob.impex

```
UPDATE GenericItem[processor=de.Hybris.platform.commerceservices.impex.
impl.ConfigPropertyImportProcessor];pk[unique=true]

$activateTriggers=$config-worldpayPaymentInfo.timeout.preparation.
triggers.activate

INSERT_UPDATE CronJob;code[unique=true];job(code);sessionLanguage
(isocode)[default=en]
;paymentInfoInquiryCronJob;paymentInfoInquiryJobPerformable;

INSERT_UPDATE Trigger;cronJob(code)[unique=true];cronExpression;
relative;active[default=$activateTriggers];maxAcceptableDelay
;paymentInfoInquiryCronJob;0 0/10 * * * ?;true;;-1
```

Riskscore and fraud

The AddOn is set up to capture and persist risk scores (from Risk Management Module or RiskGuardian) from Worldpay and display these in various business tools.

The AddOn also provides a way to change the order-flow depending on the values of the risk scores. The way this is done is to take control of the step:

fraudCheck in *order-process.xml:*

order-process.xml

```
<action id="fraudCheck" bean="fraudCheckOrderInternalAction">
   <transition name="OK" to="sendOrderPlacedNotification"/>
   <transition name="POTENTIAL" to="manualOrderCheckCSA" />
</action>
```

1. by re-aliasing the fraudCheckOrderInternalAction bean and configuring the providerName to reference Worldpay:

   worldpayapi-spring.xml

   ```
   <bean id="fraudCheckOrderInternalAction" class="com.worldpay.
   fulfilmentprocess.actions.order.
   WorldpayFraudCheckOrderInternalAction" parent="
   worldpayAbstractFraudCheckAction">
       <property name="fraudService" ref="fraudService"/>
       <property name="providerName" value="worldpay"/>
       <property name="configurationService" ref="
   configurationService"/>
   </bean>
   ```

2. By re-aliasing the fraudService so it contains only the worldpayFraudServiceProvider as a provider.

   worldpayapi-spring.xml

   ```
   <alias alias="fraudService" name="worldpayFraudService"/>
   <bean id="worldpayFraudService" class="de.Hybris.platform.fraud.
   impl.DefaultFraudService">
       <property name="providers">
           <list>
               <ref bean="worldpayFraudServiceProvider"/>
           </list>
       </property>
   </bean>
   ```

3. The worldpayFraudServiceProvider has a list of symptoms:

   worldpayapi-spring.xml

```
<bean id="worldpayFraudServiceProvider" class="de.Hybris.platform.
fraud.impl.DefaultHybrisFraudServiceProvider">
    <property name="providerName" value="worldpay"/>
    <property name="symptomList">
        <list>
            <ref bean="worldpayRiskScoreFraudSymptom"/>
            <ref bean="worldpayRiskGuardianFraudSymptom"/>
        </list>
    </property>
</bean>
```

4. The symptoms defined are:

   WorldpayRiskGuardianFraudSymptom takes care of symptoms from the worldpayRiskGuardian system and worldpayRiskScoreFraudSymptom takes care of the default returned risk score value.

   WorldpayFraudCheckOrderInternalAction will take a configuration value that can be updated in *local.properties.*

   worldpayapi-spring.xml

```
<bean id="worldpayRiskGuardianFraudSymptom" class="com.worldpay.
fraud.symptoms.WorldpayRiskGuardianFraudSymptom">
    <property name="symptomName" value="
WorldpayRiskGuardianFraudSymptom"/>
</bean>

<bean id="worldpayRiskScoreFraudSymptom" class="com.worldpay.fraud.
symptoms.WorldpayRiskScoreFraudSymptom">
    <property name="symptomName" value="
WorldpayRiskValueFraudSymptom" />
</bean>
```

5. The threshold to mark an order as fraudulent is defined as:

   worldpayapi/project.properties.template

```
worldpayapi.fraud.scoreLimit=80
```

These customisations are easily configured by adding or removing symptoms to the worldpayFraudServiceProvider or modifying the value of the scoreLimit used in **WorldpayFraudCheckOrderInternalAction.**

Deletion of saved cards due to fraud

When an order placed with a saved card is rejected manually by a customer support agent due to fraud suspicions, the saved payment info (saved card) used in the transaction is deleted from the user's profile so it cannot be used again in subsequent orders. This behaviour can be customised by realiasing the bean:

```
    <alias name="worldpayOrderManualCheckedAction" alias="
orderManualCheckedAction"/>
    <bean id="worldpayOrderManualCheckedAction" class="com.worldpay.
fulfilmentprocess.actions.order.WorldpayOrderManualCheckedAction"
parent="worldpayAbstractOrderAction" />
```

Customer IP Address

Another fraud information used by Worldpay is the customer's IP address. In the redirect flow, the IP is captured by Worldpay so it does not need to be sent in the request XML. In the direct / CSE flow, the correct customer IP needs to be sent in the request XML. As the Hybris installation may be set up behind a load balancer that transfers the customer's IP address to a header attribute and replaces the address of the request to that of the load balancer. As the header attribute name depends on the load balancer in place, a strategy is available to retrieve the customer IP from the configured header:

```
<alias name="defaultWorldpayCustomerIpAddressStrategy" alias="
worldpayCustomerIpAddressStrategy"/>
<bean id="defaultWorldpayCustomerIpAddressStrategy" class="com.worldpay.
strategy.impl.DefaultWorldpayCustomerIpAddressStrategy">
    <property name="headerName" value="X-Forwarded-For"/>

    <!-- Possible headers that may contain the customer IP if load
balancer is
        <property name="headerName" value="Proxy-Client-IP"/>
        <property name="headerName" value="WL-Proxy-Client-IP"/>
        <property name="headerName" value="HTTP_CLIENT_IP"/>
        <property name="headerName" value="HTTP_X_FORWARDED_FOR"/>
    -->
</bean>
```

Notification commands

The AddOn is set up to accept all Worldpay notification commands, but only processes **"AUTHORISED"**, **"CAPTURED"**, **"REFUSED"**, **"CANCELLED"**, **"REFUNDED"**, **"SETTLED"** and **"REFUND_WEBFORM_ISSUED"** (depending on the chosen recipe)**.**

- **wp_b2c_acc ,wp_b2b_acc**: AUTHORISED, CAPTURED, REFUSED, CANCELLED.
- **wp_b2c_acc_oms**: AUTHORISED, CAPTURED, REFUSED, CANCELLED, REFUNDED, SETTLED, REFUND_WEBFORM_ISSUED.

To process updates for other messages you will need to extend the method **processOrderNotificationMessage** in the class **DefaultOrderNotificationService**.

It is recommended to do this by adding a dependency to the **worldpayapi** in your own project extension and re-aliasing the **orderNotificationService** spring bean like so:

```
```

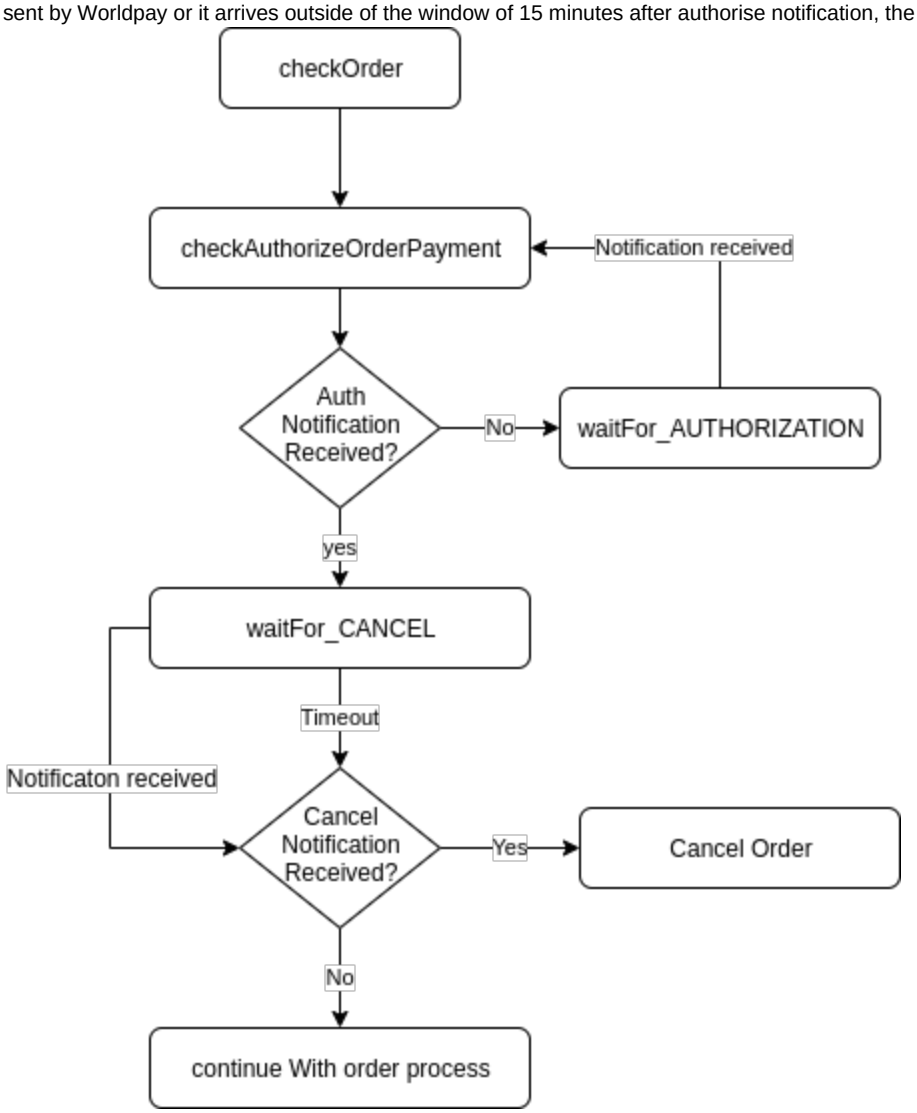
```
<alias alias="orderNotificationService" name="
myOrderNotificationService"/>
<bean id="myOrderNotificationService" class="com.myproject.core.
services.impl.DefaultOrderNotificationService" parent="
defaultOrderNotificationService" />
```

Cancel notification

Each time an order is placed and the payment has been completed, on Worldpay side there is a risk measurement which get a risk score rejection. If the risk is to high a notification is sent from Worldpay to Hybris to let us know the cancel action. When the cancel notification is

received on Hybris it is processed and depending on the order status it will processed or discarded.

There is an step on the order process waiting for Worldpay cancel notification with a timeout of 15 minutes. If the cancel notification arrives after the authorisation in a window of 15 minutes, the notification will be processed and the order will be cancelled, but in case of the notification is not sent by Worldpay or it arrives outside of the window of 15 minutes after authorise notification, the order will be processed as usual.



For customising the default cancel action, it would be necessary to implement your own CancelWholeOrderDueToCancelNotificationStrategy and re-aliasing the following bean from worldpayapi extension:

```
    <alias name="
defaultWorldpayCancelWholeOrderDueToCancelNotificationStrategy" alias="
cancelWholeOrderDueToCancelNotificationStrategy"/>
    <bean id="
defaultWorldpayCancelWholeOrderDueToCancelNotificationStrategy"
          class="com.worldpay.orderprocess.strategies.cancel.impl.
DefaultWorldpayCancelWholeOrderDueToCancelNotificationStrategy">
        <constructor-arg name="worldpayPaymentTransactionService" ref="
worldpayPaymentTransactionService"/>
        <constructor-arg name="modelService" ref="modelService"/>
    </bean>
```

Placing orders from notifications

In order to avoid reconciliation mismatch or missed orders due to possible errors in the redirections on the HOP or iframe checkout flow, the plugin has a functionality that when receiving a server to server notification for "**AUTHORISED**" and there is no payment transaction in a matching cart with the Worldpay order code, the *DefaultWorldpayPlaceOrderFromNotificationStrategy* will place an order. The recommendation is to adapt the provided strategy to the custom needs when placing orders by creating a new implementation of the *WorldpayPlaceOrderFromNotificationStrategy* interface and realiasing the bean:

```
<bean id="worldpayPlaceOrderFromNotificationStrategy" class="com.
worldpay.strategies.impl.
DefaultWorldpayPlaceOrderFromNotificationStrategy">
```

> In order to test this functionality to check the new implementation, one of the easiest ways is not finishing the HOP or iframe flows, and send a mocked notification from the /worldpayresponsemock/responses test page.

The plugin handles the scenario when an order is already placed after redirection from HOP or iframe and will redirect the customer to the order confirmation page.

Cancellations

Introduction

To Void a payment transaction (reversal with credit cards) a separate void process has been defined, to take care of cancelling orders at Worldpay. This process is started when the CancelFinishedEvent is triggered. This can, among other places, be done by clicking the "Cancel Order" button in the Customer Support Backoffice Perspective.

The process is defined as follows:

worldpay-void-process.xml

```
<?xml version="1.0" encoding="utf-8"?>
<process xmlns="http://www.Hybris.de/xsd/processdefinition" start="
sendVoidCommand" name="worldpay-void-process" processClass="com.
```

```
worldpay.voidprocess.model.WorldpayVoidProcessModel">

    <action id="sendVoidCommand" bean="worldpayCancelOrderAction">
        <transition name="OK" to="end"/>
        <transition name="NOK" to="waitForManualRetry"/>
    </action>

    <wait id="waitForManualRetry" then="sendVoidCommand">
        <event>${process.code}_VOID_PAYMENT</event>
    </wait>

    <end id="end" state="SUCCEEDED">Success</end>

</process>
```

There is only one action in this process, which will send a **CancelOrRefund** request to Worldpay. Further logic can be added by customising the process.

The worldpay-void-process is started by the *ImmediateCancelRequestExecutor* through the *OrderCancelNotificationServiceAdapter* interface, which is implemented by *WorldpayOrderCancelNotificationServiceAdapter,* which publishes the CancelFinishedEvent. The process is only triggered for complete cancellations of orders.

The bean *immediateCancelRequestExecutor* has been aliased by *worldpayImmediateCancelRequestExecutor* in the worldpayfulfilment extension:

worldpayfulfilment-spring.xml

```
<alias name="worldpayImmediateCancelRequestExecutor" alias="
immediateCancelRequestExecutor"/>
<bean id="worldpayImmediateCancelRequestExecutor" class="de.Hybris.
platform.ordercancel.impl.executors.ImmediateCancelRequestExecutor"
      scope="prototype">
    <property name="modelService" ref="modelService"/>
    <property name="orderCancelRecordsHandler" ref="
orderCancelRecordsHandler"/>
    <property name="completeCancelStatusChangeStrategy" ref="
setCancellledStrategy"/>
    <property name="notificationServiceAdapter" ref="
worldpayOrderCancelNotificationServiceAdapter"/>
</bean>
```

Please note that only a "notificationServiceAdapter" is provided by the AddOn. From Hybris documentation in Order cancel service:

OrderCancelPaymentServiceAdapter

This interface is used to recalculate an order after the cancel operation has been finished. It is used by classes: **ImmediateCancelRequest Executor**, **WarehouseResponseExecutor**. Order Cancel Service does not provide a default implementation of this interface. Users should supply their own implementation and plug it in using Spring configuration for the aforementioned classes. If not provided, orders are not automatically recalculated by Order Cancel Service after cancel operation is finished (order recalculation can be then performed externally to Order Cancel Service)

OrderCancelWarehouseAdapter

This interface is used to forward cancel requests to a warehouse. It is used by class: **WarehouseProcessingCancelRequestExecutor**. A mock implementation (**DefaultWarehouseAdapterMock**) is provided by default. Users should supply their own implementation and plug it in using Spring configuration for the aforementioned class.

Cancellation of captured orders

Hybris does not allow cancellation of captured orders. In order to completely avoid wrongful cancellations of captured orders, the worldpayOrderCancelDenialStrategy has been added to the candelDenialStrategies list of defaultOrderCancelService:

worldpayapi-spring.xml

```
<bean id="worldpayOrderCancelDenialStrategy" class="com.worldpay.
ordercancel.impl.denialstrategies.WorldpayOrderCancelDenialStrategy">
    <property name="reason">
        <bean class="de.hybris.platform.ordercancel.
DefaultOrderCancelDenialReason">
            <property name="code" value="4"/>
            <property name="description" value="Order cannot be
cancelled as there are captured transaction entries."/>
        </bean>
    </property>
</bean>

<bean id="worldpayCancelOrderServiceListMergeDirective" depends-on="
defaultOrderCancelService" parent="listMergeDirective">
    <property name="add" ref="worldpayOrderCancelDenialStrategy"/>
    <property name="listPropertyDescriptor" value="
cancelDenialStrategies" />
</bean>
```

Cancellation of APM's - *B2C only*

As APM's are auto-captured in Worldpay (except Klarna), it is not possible to cancel any order containing an APM payment transaction. This consequently means that out of the box with our AddOn, orders won't be cancellable until the authorised notification has been processed. This is because the system won't have information about the payment type before this time.

This has been implemented by adding the worldpayApmOrderCancelDenialStrategy to the candelDenialStrategies list of defaultOrderCancelService:

worldpayapi-spring.xml

```
<bean id="worldpayApmOrderCancelDenialStrategy" class="com.worldpay.
ordercancel.impl.denialstrategies.WorldpayApmOrderCancelDenialStrategy">
    <property name="reason">
        <bean class="de.Hybris.platform.ordercancel.
DefaultOrderCancelDenialReason">
            <property name="code" value="5"/>
            <property name="description" value="Order cannot be
cancelled as payment was made through an APM or is still unknown."/>
        </bean>
    </property>
```

```
</bean>

<bean id="worldpayApmCancelOrderServiceListMergeDirective" depends-on="
defaultOrderCancelService" parent="listMergeDirective">
    <property name="add" ref="worldpayApmOrderCancelDenialStrategy"/>
    <property name="listPropertyDescriptor" value="
cancelDenialStrategies" />
</bean>
```

This can, of course, be changed on a project, but a process has to be set up to handle void of payments in the event of cancellations happening. In the event of the customer using an APM, you want to be sure that the customer is aware the order is cancelled as they may already have actioned payment.

Backoffice support

The AddOn customises the backoffice customersupport perspective by including the action:

worldpaycancelorderaction

- Overrides the Hybris out of the box cancelorderaction by offering customised business rules.

Refunds

The AddOn provides refunds functionality integrated with a customised return-process, explained in Return Process.

Backoffice support

The AddOn customises the backoffice customersupport perspective by including the action:

worldpaycreatereturnrequestaction

- Overrides the Hybris out of the box create return request action by offering different business rules regarding return requests for orders paid with:
- - Card: A return request can only be created when the order transaction contains a non-pending CAPTURE transaction entry.
    - APM: A return request can only be created if the APM supports automatic refunds and the order transaction contains a non-pending SETTLED transaction entry.

Checkout Flow

Checkout flows are used by the Hybris Accelerator to control the screen web flow.

The default checkout group holds the information about different checkout steps, validation results and progress bar configuration.

The new checkout flows defined in **worldpayaddon** provide customised payment, HOP and summary steps for B2C (adaptive and responsive). The **worldpayB2CCheckoutGroup** and **worldpayB2CResponsiveCheckoutGroup** are defined in *b2c-multi-step-checkout-spring.xml*. The checkout process can be adapted on project-basis requirements by modifying the flow defined in the beans mentioned. The step definitions, validators, validation results and redirects in the checkout flow can also be found in the corresponding configuration files.

The new checkout flows defined in **worldpayb2baddon** provide customised payment and summary steps for B2B (responsive). The **worldpayB2BCheckoutGroup**is defined in b2b-multi-step-checkout-spring.xml. The checkout process can be adapted on project-basis requirements by modifying the flow defined in the beans mentioned. The step definitions, validators, validation results and redirects in the checkout flow can also be found in the corresponding configuration files.

Storefronts are generally mapped with a checkout group, based on checkout steps and flows they require. By default, storefronts generated from the template extension will be using **defaultCheckoutGroup**. Whenever a new checkout flow group is created, it is required to map it to desired base stores. This can be achieved by running the store.impex script manually or by a system update with project data for selected basestores:

Customising the Worldpay AddOn

Order Code generation strategy

To use a different order code strategy you can simply re-alias the worldpayGenerateMerchantTransactionCodeStrategy bean, below is the default:

worlpayapi-spring.xml

```
<alias name="defaultWorldpayGenerateMerchantTransactionCodeStrategy"
alias="worldpayGenerateMerchantTransactionCodeStrategy" />
<bean id="defaultWorldpayGenerateMerchantTransactionCodeStrategy"
class="com.worldpay.core.services.strategies.impl.
WorldPayGenerateMerchantTransactionCodeStrategy"/>
```

The customisation is not part of the supplied AddOn. If any customisation is required by the integrator, it can be achieved by extending the default Hybris implementation of the **PaymentService,** in *de.Hybris.platform.payment.impl.DefaultPaymentServiceImpl* and the method *getNewPaymentTransactionEntryCode.*

Authenticated Shopper Id strategy

The AuthenticatedShopperId is a required field by Worldpay when using and creating tokens. The AddOn supplies the *worldpayAuthenticatedShopperIdStrategy* for this purpose, and by default, the CustomerId is used. If the CustomerId is not set, the originalUID is used instead.

To use a different authenticated shopper Id strategy you can simply re-alias the worldpayAuthenticatedShopperIdStrategy bean, below is the default:

worldpayapi-spring.xml

```
<alias name="defaultWorldpayAuthenticatedShopperIdStrategy" alias="
worldpayAuthenticatedShopperIdStrategy"/>
<bean id="defaultWorldpayAuthenticatedShopperIdStrategy" class="com.
worldpay.strategy.impl.
DefaultWorldpayAuthenticatedShopperIdUsingCustomerIdStrategy"/>
```

Token Event Reference Creation strategy

To use a different token event reference creation strategy you can simply re-alias the worldpayTokenEventReferenceCreationStrategy bean, below is the default:

worldpayapi-spring.xml

```
<alias name="defaultWorldpayTokenEventReferenceCreationStrategy" alias="
worldpayTokenEventReferenceCreationStrategy"/>
<bean id="defaultWorldpayTokenEventReferenceCreationStrategy" class="
com.worldpay.service.payment.impl.
DefaultWorldpayTokenEventReferenceCreationStrategy">
    <property name="cartService" ref="cartService"/>
</bean>
```

Add Shipping tracking information to consignments strategy

On Klarna payments, on capture command, it is recommendable to send the tracking Ids of the order consignments, for doing so, you just need to implement your own logic for adding these info. The following strategy is injected on the addShippingTrackingInfoAction of the consignment-process.xml.

To use a different logic for enrich the consignments with the shipping tracking information you can simply re-alias the addShippingTrackingInfoToConsignmentStrategy bean, below is the default:

worldpayfulfilment-spring.xml

```
<alias name="
defaultWorldpayAddShippingTrackingInfoToConsignmentStrategy" alias="
addShippingTrackingInfoToConsignmentStrategy"/>
<bean id="defaultWorldpayAddShippingTrackingInfoToConsignmentStrategy"
class="com.worldpay.consignmentprocess.strategies.consigment.impl.
DefaultWorldpayAddShippingTrackingInfoToConsignmentStrategy"/>
```

Modifying Javascript business logic for ApplePay

To use a different business logic for ApplePay you must modify in your `local.properties` file the value for the key `worldpayaddon.javascript.paths.responsive` removing the reference to `worldpayApplePay.js`. By doing so, you will be removing the javascript injection created by the addon, hence you will need to develop the business logic of the js as you please and inject it to your storefront.

Modifying Javascript payment flow behaviour for GooglePay and ApplePay

There are two methods in the `worldpayaddon.js,` the `PaymentMethodChange` and `PaymentFlow` which can be changed in case of any customisation needed on the behaviour when the payment method has changed or you need to add any other logic on the reset payment flow.

Integration with Klarna

Limitations:

Klarna is an Alternative Payment Method that allows instant payment, deferred payment or sliced payment. The integration with Klarna has some differences from the integration with other payment methods. Klarna needs to receive from the merchant the tax applied to the order, the order lines of the cart, the shipping cost, the order level discount and so on, plus the shipping track info on capture payment during the order process. The implementation of the integration with Worldpay in the Hybris AddOn covers, as an example, orders with order level promotion with a percentage and absolute discounts applied. This means it will prorate the percentage discount across all the items in the order and will add a Discount order line entry with the absolute discount amount.

Front-end integration:

The AddON provides direct integration with Klarna through Worldpay. Once the customer chooses to pay with Klarna, the user is shown the HTML content provided by Worldpay. This content could be displayed on a full page, or in an iframe or as content in the payment page. Once the order is placed in Klarna, the customer is redirected to a provided endpoint back to the Hybris installation. The AddOn inquiries the status of the order, and the response contains HTML that must be shown to the customer. This content could also be displayed on a page, an iframe or as content on the current page. The provided implementation displays the content on a page and should be adapted to the business requirements.

Country restrictions:

Klarna only works in some specific countries with some specific currencies per country. The integration with Klarna also needs the current customer locale. The accelerator does not support Finland or Sweden (for instance) as it supports the UK or Germany. This means the locale set up for the customer in the UK Apparel or DE Apparel stores are gb-en and de-de, but changing the shipping country to Finland will not change the locale to fi-fi. The integration must be amended to comply with the following validations done in Worldpay Gateway:

http://support.worldpay.com/support/kb/gg/klarna/klarna.htm#topics/xml_input_examples_-_validations.htm%3FTocPath%3DXML%2520input%2520examples|_____3

Shipping costs:

In the implementation of the integration only the total shipping costs are sent to Klarna, it is not taking in consideration the shipping cost of individual consignments (in the event of a Marketplace integration, for instance).

Klarna Strategy:

There is a strategy available so that the implementation of the creation of the order lines sent to Klarna can be changed:

worldpayapi-spring.xml

```
<alias name="defaultWorldpayKlarnaStrategy" alias="
worldpayKlarnaStrategy"/>
<bean id="defaultWorldpayKlarnaStrategy" class="com.worldpay.service.
payment.impl.DefaultWorldpayKlarnaStrategy">
    <property name="commonI18NService" ref="commonI18NService"/>
        <property name="worldpayUrlService" ref="worldpayUrlService"/>
</bean>
```

Smartedit support

The AddOn makes available the *WorldpayPaymentPage* that defines the payment flow into SmartEdit. But changing the template is not supported by SAP Hybris in the 6.6 version.

In order to add more types of pages so they are available in SmartEdit:

worldpayapi-spring.xml

```
<bean id="pageModelPopulatingConvertersMapMergeDirective" depends-on="
pageModelConverterFactory" parent="mapMergeDirective">
        <property name="key">
            <value type="java.lang.Class">com.worldpay.model.
WorldpayPaymentPageModel</value>
    </property>
    <property name="value" ref="contentPageModelConverter"/>
</bean>

<alias name="worldpaySupportedPagesSetFactoryBean" alias="
cmsSupportedPages" />
<bean id="worldpaySupportedPagesSetFactoryBean" parent="
defaultCmsSupportedPages">
    <property name="sourceSet">
        <set merge="true">
            <value type="java.lang.Class">com.worldpay.model.
WorldpayPaymentPageModel</value>
        </set>
    </property>
</bean>
```

Dynamic interaction support

The AddOn supports the use of the field *dynamicInteractionType* to adapt the shopper interaction based on the method a transaction connects to Worldpay for only **direct payment requests** removing the need of having several merchants to differentiate the orders placed in the e-commerce or ASM.

Interaction types currently supported by Worldpay:

- ECOMMERCE: For transactions through the storefront placed by the clients
    - MOTO: For mail or telephone orders placed using ASM

GPDR

With the release of SAP Hybris 6.6, and several functionalities were added:

- Personal Data Erasure: Right to be forgotten from a website
    - Generic Audit: Given a moment of time, the user can ask for every information recorded about him/her in the system.

The AddOn provides functionalities to request the deletion of a token from Worldpay when the user requests the deletion of their account via *WorldpayPaymentInfoRemoveInterceptor*.

The file *y-ext/ext-worldpay/worldpayapi/resources/worldpayapi-customerdata-audit.xml* provides an extension to the Out of the Box PersonalDataReport and includes information that is extended by the plugin.

Mac Validation

The plugin has support for both MD5 and HMAC256 algorithms to calculate the MAC.

```
    <alias name="md5MacValidator" alias="macValidator"/>
    <bean id="md5MacValidator" class="com.worldpay.service.mac.impl.
MD5MacValidator"/>
    <bean id="hmac256MacValidator" class="com.worldpay.service.mac.impl.
HMAC256MacValidator"/>
```

By default, md5MacValidator is set up, but due to project necessities and configuration, changing the alias from *md5MacValidator* to *hmac256MacValidator* will change how the validation is performed.

In order to configure it depending on environments, you can use 'spring.profiles.active' and restrict a bean to a specific profile.

3DS2 / 3DS-Flex Adding Risk Data

During 3DS flow, you can provide Worldpay with additional information in the `<riskData>` element to increase the chances that the shopper won't be challenged. For doing so, we need to override the bean defaultWorldpayRiskDataService, which method createRiskData is responsible of adding the <riskData> information.

1. Create your own RiskDataService. In order to accomplish it, you just need to implement the com.worldpay.service.payment.WorldpayRiskDataService interface, overriding any of the following methods:
    a. createTransactionRiskData
    b. createShopperAccountRiskData
    c. createAuthenticationRiskData
2. It is mandatory to override the createRiskData which creates the risk data object for the initial payment request.
3. Declare your bean in you spring.xml on your extension, overriding the alias worldpayRiskDataService.
4. Make your extension dependant of worldpayapi extension in the extensioninfo.xml

Project Customisation Points of Consideration

The Worldpay AddOn is an accelerator for your project, however, it's important to understand exactly what is delivered for your project scope.

Supported Payment Flows: HOP (Iframe and Full page redirect) (B2C only), CSE and saved card using tokenisation.

Supported Checkout Flows: Registered / Guest

Fulfilment Processes: OOTB Fulfilment and OMS

Worldpay order notification messages persisted and processed via Cron Jobs

Accelerator: SAP Commerce 2005

Fraud: Data Capture and Manual Fraud workflow

Database : hSQL / MySQL 5.7 tested

Supported Environments

The AddOn has been developed and tested with the following versions and databases:

- Hybris 2005
- Java 11
- MySQL
- HSQL

Supported Browsers:

Microsoft Internet Explorer	10, 11	Backoffice and Cockpits only
Microsoft Edge	Evergreen	
Mozilla Firefox	Evergreen	
Google Chrome	Evergreen	
Apple Safari	Evergreen	