

1. Worldpay SAP Commerce Cloud 2105 Connector R4.0	2
1.1 SAP Commerce Cloud 2105 Connector R4.0 - FAQ / troubleshooting	2
1.2 SAP Commerce Cloud 2105 Connector R4.0 - Documentation	5
1.2.1 SAP Commerce 2105 Connector R4.0 - Implementation Guide	6
1.2.1.1 Testing Apple Pay 2105 R4.0	47
1.2.1.2 Setting up Apple Pay Merchant Certificates 2105 R4.0	47
1.2.2 SAP Commerce 2105 Connector R4.0 - OCC Technical Implementation Guide	48
1.2.3 SAP Commerce 2105 Connector R4.0 - Technical Implementation Guide	59
1.2.4 SAP Commerce 2105 Connector R4.0 - B2B Technical Implementation Guide	75

Worldpay SAP Commerce Cloud 2105 Connector R4.0

This section documents information and documentation shared with Worldpay for the connector of SAP Commerce 2105.

- [SAP Commerce Cloud 2105 Connector R4.0 - FAQ / troubleshooting](#)
- [SAP Commerce Cloud 2105 Connector R4.0 - Documentation](#)

Search this release

Release Notes

Features:

- Guaranteed Payments
- 3DS changed to 3DS v2

Bug fixes:

- Order Confirmation page is missing information in different APMs in Spartacus
- CSE cards' payment failing with 400 HTTP error

Tag Release

https://github.com/Worldpay/hybris/releases/tag/release_2105_4.0

SAP Commerce Cloud 2105 Connector R4.0 - FAQ / troubleshooting

To gain knowledge and understanding of the workings of the Worldpay Connector for SAP Commerce Cloud, we recommend installing it on a fresh install of SAP Commerce Cloud with only the b2caccelerator.

This can easily be done by using one of the recipes provided with the Connector.

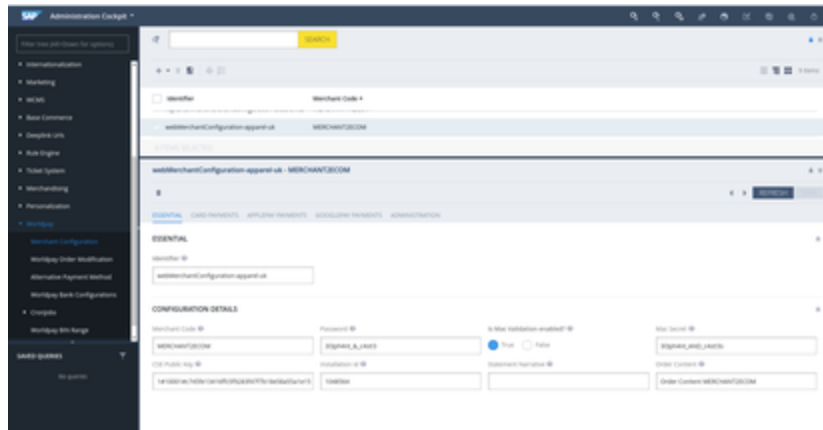
Pre-Go Live Checks

Topics to consider

- Request your merchant codes on an early stage of the project.
- In case the project needs pick up in-store, please ensure that all available stores have a postal code.
- Request the activation of "Dynamic interaction type" when setting up the merchants.
- If using CSE, remember to generate your own public key and update it in the merchants.xml file.
- Check the production URLs for 3ds2 Flex with your Worldpay manager.

Checklist when facing an issue.

1. Carefully read through all the documentation supplied with the plugin.
2. Ensure the version of the plugin is corresponding to your hybris version.
3. Carefully follow the implementation guide and verify that the desired functionalities are installed correctly.
4. Follow the step by step guide below to verify, that the merchant configurations in SAP Commerce Backoffice matches the configuration in the Worldpay merchant interface.
 - a. To check your configured merchants:
 - i. Go to the SAP Commerce Backoffice
 - ii. Go to Worldpay Merchant Configuration
 - iii. The list view will display different merchant configuration per channel ("web", "mobile" or "customerService")
 - iv. Clicking on a merchant, all it's properties will be displayed

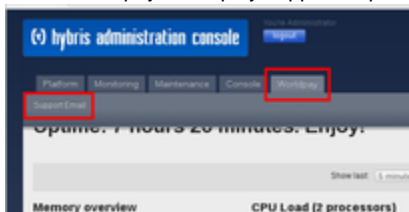


5. Ensure that the communication with Worldpay (including receiving order modifications) is not in any way blocked or modified by your local infrastructure.
 - a. Be mindful that your hybris nodes have to be able to reach the Worldpay test or production system.
 - b. The node which has the worldpaynotifications extension installed has to be reachable from the Worldpay test or production system.
6. Check API requests and responses. They are saved in the order and can be viewed in backoffice or through OData.
 - a. To display them in Backoffice:
 - i. Go to Backoffice
 - ii. Go to Order Orders
 - iii. Select an Order
 - iv. Click on the Worldpay tab
 - v. Payloads will be displayed under the "Api Request/Response Payloads" section

Raising a request

When raising a support request, please follow the steps below.

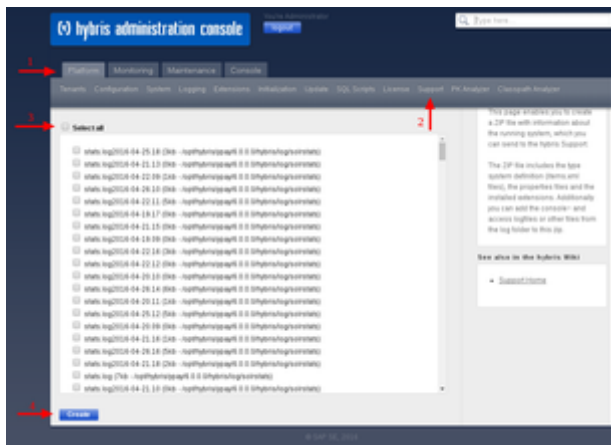
1. Sending the support email to Worldpay.
 - a. Set up the support email functionality as described in the Implementation guide - configure the support email.
 - b. Open HAC
 - c. Select Worldpay Wordplay support. A preview of email body and a button to send support email is displayed.



- d. Click send email button to send a support email to Worldpay. When the email is sent a message showing receiver email is displayed below the button.



2. Providing a detailed description of the error you're getting
 - a. Include the whole stack trace if available.
 - b. Include the full log of the communication with Worldpay (by default outgoing and incoming messages are printed to the tomcat log in `$(HYBRIS_LOG_DIR)/tomcat/console*`)
 - c. Include screenshots of the error (of the whole browser - including the address-field).
 - d. Include the hybris support zip. The zip can be obtained by following the steps below
 - i. Go to the hybris Administration Console (HAC)
 - ii. Go to Platform Support
 - iii. Click "Select all" and download the zip.



To gain knowledge and understanding of the workings of the Worldpay Connector for SAP Commerce Cloud, we recommend installing it on a fresh install of SAP Commerce Cloud with only the b2caccelerator. This can easily be done by using one of the recipes provided with the Connector.

Pre-Go Live Checks

Topics to consider

- Request your merchant codes on an early stage of the project.
- In case the project needs pick up in-store, please ensure that all available stores have a postal code.
- Request the activation of "Dynamic interaction type" when setting up the merchants.
- If using CSE, remember to generate your own public key and update it in the merchants.xml file.
- Check the production URLs for 3ds2 Flex with your Worldpay manager.

Checklist when facing an issue.

1. Carefully read through all the documentation supplied with the plugin.
2. Ensure the version of the plugin is corresponding to your hybris version.
3. Carefully follow the implementation guide and verify that the desired functionalities are installed correctly.
4. Follow the step by step guide below to verify, that the merchant configurations in hybris (merchants.xml) matches the configuration in the Worldpay merchant interface.
 - a. To check your configured merchants:
 - i. Go to the Hybris Administration Console (HAC)
 - ii. Go to Console Scripting Languages
 - iii. Execute statements of the pattern: "worldpayMerchantConfiguration.[channel].[property]" where:
 1. [channel] can be: "web", "mobile" or "customerService"
 2. [property] can be "code", "password", "macValidation", "macSecret", "csePublicKey", "installationId", "statementNarrative" or "orderContent"

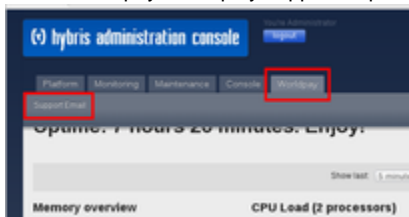


5. Ensure that the communication with Worldpay (including receiving order modifications) is not in any way blocked or modified by your local infrastructure.
 - a. Be mindful that your hybris nodes have to be able to reach the Worldpay test or production system.
 - b. The node which has the worldpaynotifications extension installed has to be reachable from the Worldpay test or production system.
6. Check API requests and responses. They are saved in the order and can be viewed in backoffice or through OData.
 - a. To display them in Backoffice:
 - i. Go to Backoffice
 - ii. Go to Order Orders
 - iii. Select an Order
 - iv. Click on the Worldpay tab
 - v. Payloads will be displayed under the "Api Request/Response Payloads" section

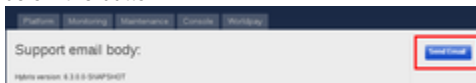
Raising a request

When raising a support request, please follow the steps below.

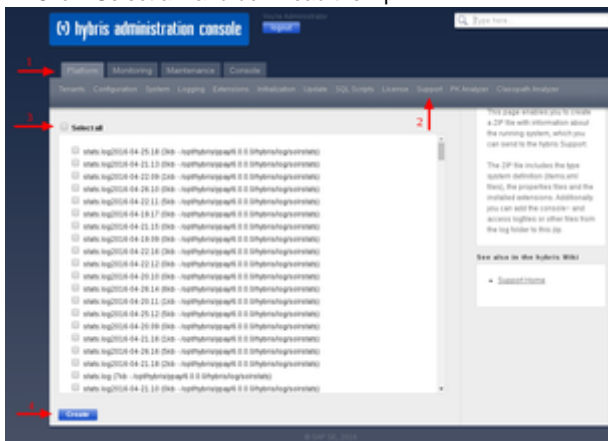
1. Sending the support email to Worldpay.
 - a. Set up the support email functionality as described in the Implementation guide - configure the support email.
 - b. Open HAC
 - c. Select Worldpay Wordplay support. A preview of email body and a button to send support email is displayed.



- d. Click send email button to send a support email to Worldpay. When the email is sent a message showing receiver email is displayed below the button.



2. Providing a detailed description of the error you're getting
 - a. Include the whole stack trace if available.
 - b. Include the full log of the communication with Worldpay (by default outgoing and incoming messages are printed to the tomcat log in `${HYBRIS_LOG_DIR}/tomcat/console*`)
 - c. Include screenshots of the error (of the whole browser - including the address-field).
 - d. Include the hybris support zip. The zip can be obtained by following the steps below
 - i. Go to the hybris Administration Console (HAC)
 - ii. Go to Platform Support
 - iii. Click "Select all" and download the zip.



- [SAP Commerce 2105 Connector R4.0 - Implementation Guide](#)
- [SAP Commerce 2105 Connector R4.0 - OCC Technical Implementation Guide](#)
- [SAP Commerce 2105 Connector R4.0 - Technical Implementation Guide](#)
- [SAP Commerce 2105 Connector R4.0 - B2B Technical Implementation Guide](#)

Reader's guide

This documentation 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 B2C sections that are not storefront specific also apply to the OCC addon. The OCC addon is documented in a separate section.

SAP Commerce 2105 Connector R4.0 - 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](#)
 - [FraudSight Configuration](#)
 - [Guaranteed Payments Configuration](#)
 - [Prime Routing Configuration](#)
 - [Level 2/3 Data](#)
 - [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](#)
 - [FraudSight](#)
 - [Level 2/3 data](#)
 - [Additional data request strategies](#)
 - [Customising the Worldpay AddOn](#)
 - [Project Customisation Points of Consideration](#)
 - [Supported Environments](#)
 - [Show used tokenised payment method on order summary step](#)

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

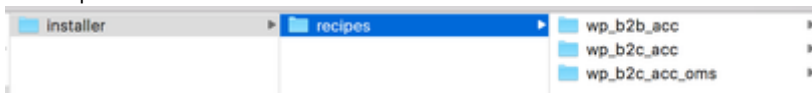
Installing the Plugin using the provided recipes

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:f

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

i 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.

i Ask your Worldpay Integration Support Manager to enable

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

i 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:

Merchant Channels (Test)						
Protocol	Active	Content	Address	Method	Response	Client Certificate
email	<input type="radio"/> yes <input checked="" type="radio"/> no	text ▼				
http	<input checked="" type="radio"/> yes <input type="radio"/> no	xml ▼	https://yoursite.com/worldpaynotifications/worldpay/merchant_callback	POST ▼	[OK]	<input type="radio"/> yes <input checked="" type="radio"/> no
shopper email	<input type="radio"/> yes <input checked="" type="radio"/> no	test uses the same messages and content settings as production				

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

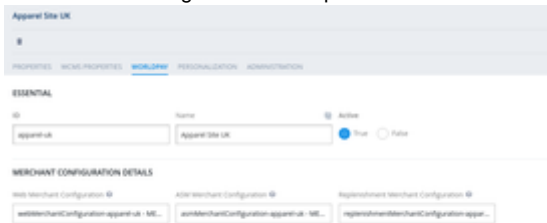
There are three types of merchant codes which can be configured:

- Web (ECOM) - Used for purchases made via the Web channel
- 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 or Telephone Order

Configuration Details

In order to configure the merchants for your site you'll need to create or select a merchant configuration for the fields: `webMerchantConfiguration`, `asmMerchantConfiguration` and `replenishmentMerchantConfiguration`.



The screenshot shows the 'Apparel Site UK' configuration page. It includes tabs for 'PROPERTIES', 'SOCIAL PROPERTIES', 'WORLDPAY', 'PERSONALISATION', and 'ADMINISTRATION'. The 'WORLDPAY' tab is active, showing 'ESSENTIAL' and 'MERCHANT CONFIGURATION DETAILS' sections. The 'ESSENTIAL' section has fields for 'ID' (apparel-uk) and 'Name' (Apparel Site UK), with an 'Active' toggle set to 'True'. The 'MERCHANT CONFIGURATION DETAILS' section has three dropdowns for 'Web Merchant Configuration', 'Assisted Merchant Configuration', and 'Replenishment Merchant Configuration', all set to 'webMerchantConfiguration-apparel-uk - ME...'.

To create a merchant configuration you'll need to define a set of fields for it.



The screenshot shows the 'Edit item webMerchantConfiguration-apparel-uk - MERCHANT2ECOM' page. It includes tabs for 'ESSENTIAL', 'CARD PAYMENTS', 'APPLEPAY PAYMENTS', 'GOOGLEPAY PAYMENTS', and 'ADMINISTRATION'. The 'ESSENTIAL' tab is active, showing 'ESSENTIAL' and 'CONFIGURATION DETAILS' sections. The 'ESSENTIAL' section has a field for 'Identifier' (webMerchantConfiguration-apparel-uk). The 'CONFIGURATION DETAILS' section has fields for 'Merchant Code' (MERCHANT2ECOM), 'Password' (30phint_8_c0e03), 'Is Mac Validation enabled?' (True), 'Max Secret' (30phint_AND_c0e03s), 'CSE Public Key' (1410001e405b13a10f5f92304707b18d8a0f1e153d873cf), 'Installation ID' (1548564), 'Statement Narrative', and 'Order Content' (Order Content MERCHANT2ECOM).

Field details:

Name	Description
code	Merchant Code
password	Merchant password set up in the Worldpay merchant interface

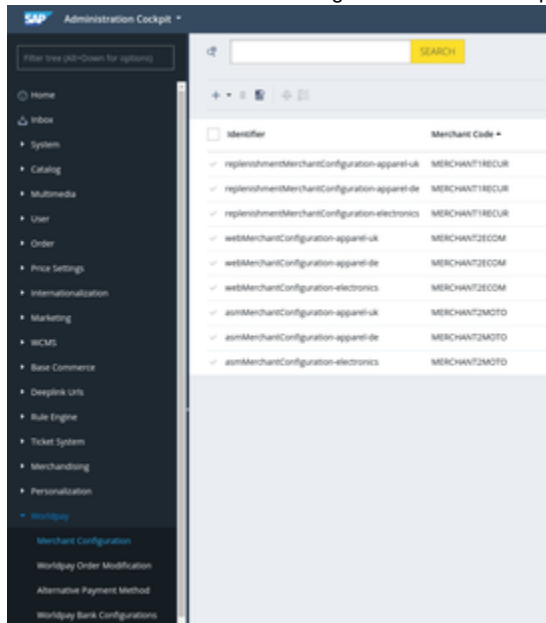
macValidation	Defines if MAC validation should be used.
macSecret	Merchant MAC (Message Authentication Code) password set up in the Worldpay merchant interface
installationId	Identification of the installation
statementNarrative	Text that is displayed on the shopper's statement.
orderContent	HTML order 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:

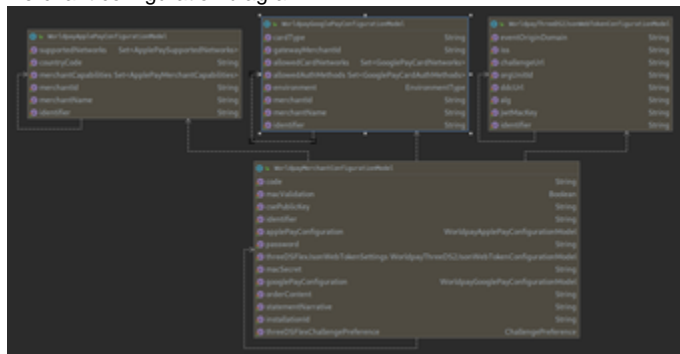
Name	Description
csePublicKey	Public key generated using the Worldpay Merchant Interface

Further explanation of these fields can be found in <https://developer.worldpay.com/docs/wpg>

You can find all the merchant configurations in the Worldpay Section in backoffice.



Merchant configuration diagram



Assisted services

If your application uses the assisted services module, you will need to configure a specific merchant and set it in the field `asmMerchantConfiguration` as shown 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 3DS2) when placing orders as an agent.

3DS2/3DS-Flex Configuration

Whether your merchant has activated 3DS or 3DS2, on the merchant configuration, Card Payments tab/ Configuration Details section, you can find the attribute `threeDSFlexJsonWebTokenSettings` which contains the 3DS-Flex configuration attributes. In the same section you'll find the attribute `3DSFlexChallengePreference` which allows you to set a preferred value for the 3DS Flex choosing from the list of values available.

The image shows two screenshots of the Worldpay configuration interface. The top screenshot is the 'Edit Item WorldpayThreeDS2JsonWebTokenConfiguration[8796093174000]' form. It contains fields for '3DS2 flex auth submit URL', 'Challenge URL', 'Comments', '3DS2 URL', 'Event Origin Domain', '3DS2 flex flow return URL', 'Identifier', '3DS ID', '3DS2 flex flow return URL', and 'Organization unit ID'. The bottom screenshot is the 'Edit Item asmMerchantConfiguration-apparel-uk - MERCHANT2NAOTO' form, specifically the 'CONFIGURATION DETAILS' section. It shows the '3DS2 Configuration' dropdown set to 'WorldpayThreeDS2JsonWebTokenConfiguration[8796093174000]' and the '3DS2 flex Challenge Preference' dropdown set to 'challengeStandard'.

The connector supports 3DS Flex with Cardinal and 3DS Flex Legacy.

For the legacy implementation check the following test values in the Worldpay documentation page <https://developer.worldpay.com/docs/wpg/directintegration/3ds2#testing>.

3DS Flex with Cardinal

Worldpay's relationship with Cardinal Commerce leverages their 3DS2 authentication service for Device Data Collection (DDC) and challenge flows.

In order to use this functionality, you must be setup for 3DS Flex and the Secure Test connection to Cardinal, before using it. For more information, contact your Worldpay Relationship Manager or Implementation Manager.

To enable the 3DS Flex with Cardinal you'll need to update the `threeDSFlexJsonWebTokenConfiguration` with the Cardinal values for the attributes : challenge URL, DDC URL and Event Domain.

The image shows a screenshot of the 'Edit Item WorldpayThreeDS2JsonWebTokenConfiguration[8796093106777]' form. The fields are populated with Cardinal values: '3DS2 flex auth submit URL' is 'ACheckoutMultiWorldpay3dssecureflex/hopresponse/authsub', 'Challenge URL' is 'https://cardinalapi.tag.cardinalcommerce.com/3DS/ChallengeSteps', 'Comments' is empty, '3DS2 URL' is 'https://cardinalapi.tag.cardinalcommerce.com/3DS/ChallengeSteps', 'Event Origin Domain' is 'https://cardinalapi.tag.cardinalcommerce.com', '3DS2 flex flow return URL' is 'ACheckoutMultiWorldpay3dssecureflex/hopresponse', 'Identifier' is 'apparel-de-threeDS2Config', and '3DS ID' is '5b0e0e4444a0e1342b940'.

For more information please refer to the worldpay documentation <https://developer.worldpay.com/docs/wpg/directintegration/cardinalsecuretest>.


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.

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 appropriate 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	Description	Notes
-----------	------	-------------	-------

		Required		
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.	
autoCancelP endingTimeo utInMinutes	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<Country Model>	No	Set of countries for which the APM is available	 The Country is related to the user's shipping address
currencies	Set<Currenc yModel>	No	Set of currencies for which the APM is available	
currencyRan ges	Set<Worldpa yCurrencyRa ngeModel>	No	Set of currency ranges for which the APM is available	
automaticRef unds	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.
bankTransfer Refunds	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	WorldpayAPMConfigur ation	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.

Secure bank transfer

This functionality is added to the bank payments in order to make the payment redirect more secure and safe. If you're offering this payment method define an encryption key for your installation.

Success URL will be encrypted using the worldpay order code and a secret encryption key

```
worldpay.orderCode.encryption.key
```

defined in the project.properties. When worldpay redirect to the success url, the controller will decrypt the order id received and validate against hybris order. If the order id is validated then the order is placed and customer is redirected to the order confirmation page, otherwise will be redirected to the choose payment method page with an error message.

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 worldpaysampledadataaddon 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 be 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	<p>The supported values for merchantCapabilities are:</p> <ul style="list-style-type: none"> • 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. <p>https://developer.apple.com/documentation/apple_pay_on_the_web/applepaypaymentrequest/1916123-merchantcapabilities</p>
supportedNetworks	<p>List of all supported card networks</p> <p>https://developer.apple.com/documentation/apple_pay_on_the_web/applepayrequest/2951831-supportednetworks</p>

- 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. The obfuscated card number and month/date will be saved in the payment info and shown in the order confirmation page.

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/googlepayAPlenable/>

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: <ul style="list-style-type: none"> • 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. 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

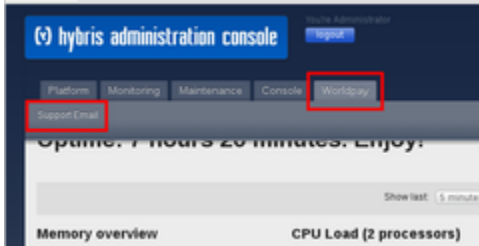
# Base Store
UPDATE BaseStore;uid[unique=true];paymentProvider;checkoutFlowGroup
;$storeId;$paymentProvider;$checkoutGroup
```

Extension properties

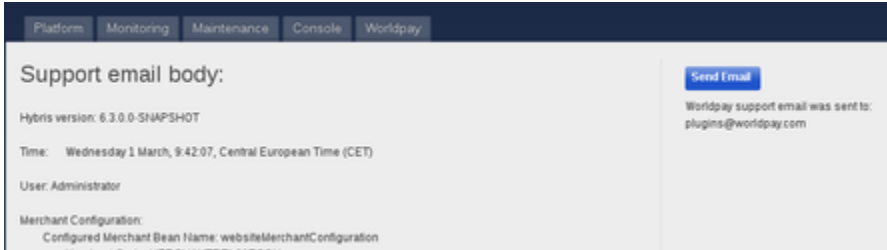
Extension	Properties	Description
worldpayresponsemock	Modification Controller Mapping	worldpayresponsemock.order.notification.endpoint should be changed in your environment specific local.properties if you want to use the mock. It has to match the mapping of the com.worldpay.worldpaynotifications.controllers.order.notification.OrderModificationController

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
<div><pre>worldpayapi.inquiry.max. number.of.retries</pre></div>	Maximum number of tries to inquiry the order status	3
<div><pre>worldpayapi.inquiry.delay. between.retries</pre></div>	Delay between retries in seconds	3

FraudSight Configuration

Worldpay provides the FraudSight solution in order to minimise creditcard fraud and increase approval rates, using a combination of data insights, technology such as machine learning, and fraud expertise.

This feature can be enabled or disabled for each website with a configuration flag in the Worlpay tab of the site.



When an order is placed with FraudSight enabled the FraudSight data from Worldpay is saved in the payment transaction entry. FraudSight data contains an id, a message, the score and risk reasons.



When FraudSight is enabled for a site, the customer needs to add the date of birth in the Payment & Billing address step in order to complete a payment.



Edit item provider=Guaranteed Payments message=ACCEPT score=1.5

Comments	Extended response
Final score	Worldpay id @
Message	Provider id
RiskGuardian id @	Guaranteed Payments
Is blocked for processing	WorldpayRiskScore score
Transaction score @	1.5
	Threshold score @
	Triggered rules
	Rule_accept

Prime Routing Configuration

Prime Routing feature is a data-driven service that examines each eligible debit transaction and routes it to the appropriate debit network, based on lowest cost. This feature can be enabled or disabled for each website with a configuration flag in the Worlpay tab of the site.

The screenshot shows the 'Prime Routing Configuration' section of the Worlpay interface. It includes a 'Name' field with the value 'electronics', an 'Active' toggle set to 'True', and a 'Is Prime Routing enabled?' toggle also set to 'True'.

Prime Routing is only enabled by Worldpay for **the US and for card payments only**. If an order is placed with Prime Routing and gets cancelled within the same US working day, the transaction will be voided and a CANCEL payment transaction entry will be created for the order.

In case the order gets cancelled after this time the transaction will be refunded and the order will have a Cancel transaction with amount 0 and a REFUND_FOLLOW_ON transaction with the all the transaction details.

The screenshot shows a transaction entry for a cancelled order. The 'Transaction' field is set to 'PaymentTransaction@PENDING'. The 'Request token' is 'WORLDWIDE@PENDING'. The 'Transaction status' is 'REJECTED'. The 'Transaction status details' are 'PROCESSOR_PENDING' and 'CANCEL'.

Level 2/3 Data

Worldpay provides the Level 2/3 Data feature so that merchants obtain lower transaction rates and a lower processing cost. This feature is **only available for US and Canadian payments and card networks**.

This feature can be enabled or disabled for each website with a configuration flag in the Worlpay tab of the site.

The screenshot shows the 'Level 2/3 Data Configuration' section of the Worlpay interface. It includes a 'Name' field with the value 'electronics', an 'Active' toggle set to 'True', and a 'Is Level 2 enabled?' toggle also set to 'True'.

Some additional configuration for level 2/3 are:

- Commodity code on products (mandatory for level 3 data)
- Card acceptor Tax Id (mandatory for both) - configured in the merchant configuration

The screenshot shows the 'Merchant Configuration' page in the Worlpay interface. It includes a 'Name' field with the value 'electronics', an 'Active' toggle set to 'True', and a 'Is Level 2 enabled?' toggle also set to 'True'.

When both flags are enabled Level 3 takes priority. The data will be validated before sending the request and if it's not valid for Level 3, it will get validated as a Level 2 request. If one of the optional fields for Level 2 is not valid that value will be removed in order to send a valid Level 2 request. When both are invalid no Level 2/3 data will be send in the request.

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
<div>typeOfPaymentTransactionToProcessSet</div>	<p>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.</p> <p>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.</p>

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

<div>daysToWaitBeforeDeletion</div>	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
<div>unpro cesse dTime InDays</div>	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" />
</bean>

<bean id="paymentTransactionDao" class="com.worldpay.dao.PaymentTransactionDao" parent="abstractPaymentTransactionDao">
    <property name="worldpayPaymentTransactionDao" ref="paymentTransactionDao" />
    <property name="worldpayMerchantService" ref="worldpayMerchantService" />
    <property name="configurationService" ref="configurationService" />
</bean>

<bean id="paymentTransactionRejectionStrategy" class="com.worldpay.cronjob.PaymentTransactionRejectionStrategy" parent="abstractPaymentTransactionRejectionStrategy">
    <!-- 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 `defaultWaitInMinutes` 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 re-aliasing 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**.

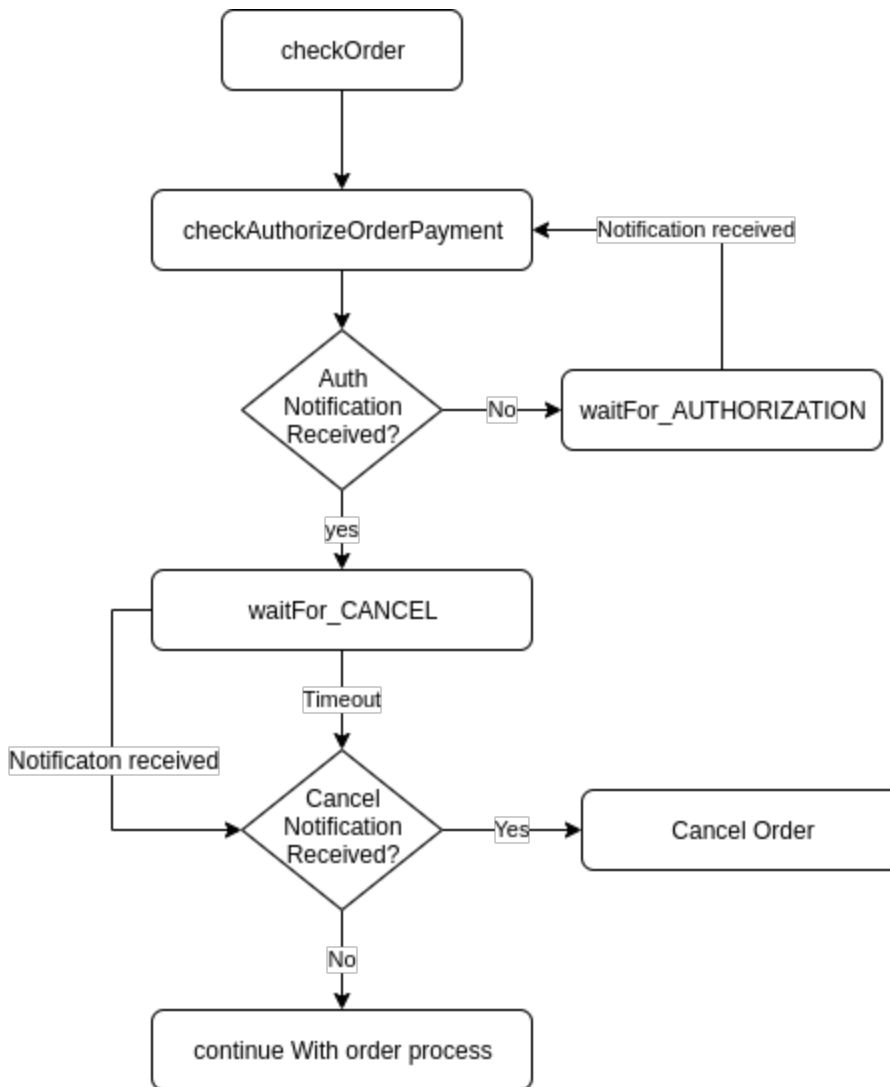
It is recommended to do this by adding a dependency to the **worldpayapi** in your own project extension and re-aliasing the **orderNotificationService** spring bean like so:

```
<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 too 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 be processed or discarded.

There is a 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 authorisation 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>

```

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 realising 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="
setCancelledStrategy" />
    <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 cancelDenialStrategies 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 `cancelDenialStrategies` 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:

FraudSight

FraudSight data is added by a strategy and in order for it to work the SI has to extend this strategy and add custom implementation for creating the custom string fields and custom numeric fields.

Extend `DefaultWorldpayFraudSightStrategy.java` and override the following methods:

```
/**
 * Creates and populates the CustomStringFields. Override this
 * method with Real values based on business requirements
 */
@Override
protected CustomStringFields createCustomStringFields() {
    throw new NotImplementedException("Implement this method based
on business requirements");
}

/**
 * Creates and populates the CustomNumericFields. Override this
 * method with Real values based on business requirements
 */
@Override
protected CustomNumericFields createCustomNumericFields() {
    throw new NotImplementedException("Implement this method based
on business requirements");
}
```


Level 2/3 data

Level 2/3 data is added by a strategy and in order for it to work the SI has to extend this strategy and add custom implementation for the methods that populate customer reference, item description and duty amount fields.

Extend `DefaultWorldpayLevel23Strategy.java` and override the following methods:

```
/**
 * Populates the customer reference. Override this method with real
 values based on business requirements
 */
@Override
protected void setCustomerReference(final AbstractOrderModel order,
final Purchase purchase) {
    throw new NotImplementedException("Implement this method based
on business requirements");
}

/**
 * Populates the product description. Override this method with
real values based on business requirements
 */
@Override
protected void setProductDescription(final ProductModel product,
final Item item) {
    throw new NotImplementedException("Implement this method based
on business requirements");
}

/**
 * Populates the duty amount. Override this method with real values
based on business requirements
 */
@Override
protected void setDutyAmount(final AbstractOrderModel order, final
Purchase purchase) {
    throw new NotImplementedException("Implement this method based
on business requirements");
}
```

Additional data request strategies

There are two list of strategies for additional data for the request defined in spring bean as following:

```
<util:list id="worldpayDirectDataRequestStrategies"
          value-type="com.worldpay.service.payment.
WorldpayAdditionalDataRequestStrategy">
    <ref bean="worldpayFraudSightStrategy"/>
    <ref bean="worldpayPrimRoutingStrategy"/>
```

```

        <ref bean="worldpayLevel23Strategy" />
    </util:list>

    <util:list id="worldpayRedirectDataRequestStrategies"
        value-type="com.worldpay.service.payment.
WorldpayAdditionalDataRequestStrategy">
        <ref bean="worldpayFraudSightStrategy" />
        <ref bean="worldpayLevel23Strategy" />
    </util:list>

```

If a new strategy is added by the SI ,in order to add it to the request needs to be added to one of the following lists:

- worldpayDirectDataRequestStrategies : for strategies applied for direct integration
- worldpayRedirectDataRequestStrategies : for strategies applied for hosted integration

Here is an example how to add a new strategy to the list as defined in OOTB hybris:

```

    <bean id="worldpayDirectDataRequestStrategiesListMergeDirective"
depends-on="worldpayDirectDataRequestStrategies" parent="
listMergeDirective">
        <property name="add" ref="newStrategyBeanToInsert" />
    </bean>

```

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:

worldpayapi-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.consignment.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 your `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	

Show used tokenised payment method on order summary step

For B2C and B2B, when the customer lands in the checkout summary step, it will see now above the subtotal section the information related to the payment used. If it is card it will see something like:

Mr. 3DS_V2_FRICTIONLESS_IDENTIFIEDVisa, 4444*****1111, 01/2022

If it is an APM, it will see something like:

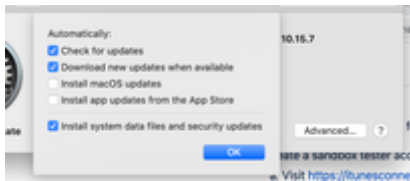
GOOGLEPAY

Testing Apple Pay 2105 R4.0

Configuring a testing account for Apple Pay requires a lot of steps that need to be followed very carefully.

1. Create a sandbox tester account
 - a. Visit <https://itunesconnect.apple.com/>
 - b. Open 'Users and Roles'
 - c. In the left menu "Sandbox" > "Testers"
 - d. Create a new user. It can't be an existing iCloud account!! Make sure the 'App store country or region' is set to United States
 - e. An email will be sent to the email address, for account validation
2. Login on a Macbook with touch bar, or a iPhone 6 or later with this new account
3. Go to "System Preferences" > "Apple ID" > "Payment & Shipping" and add shipping address in united states (for example 1 Infinite Loop, Cupertino, California 95014, United States)
4. Go to "System Preferences" > "Wallet and Apple Pay" and add card. Note that in the right corner it says "Sandbox" (example: 5204 2477 5000 1505, exp. date 11/2022, CVC: 111)
5. If the card is not added, try a card from <https://developer.apple.com/apple-pay/sandbox-testing/>
6. Verify if you can make a payment on <https://applepaydemo.apple.com/>

Common problems



Make sure you have "Install system data files and security updates" checked

If you add a card in the wallet on the iPhone, it might not sync to your account on the mac. Try adding it on the mac instead.

Setting up Apple Pay Merchant Certificates 2105 R4.0

When you register for Apple Pay you will receive a certificate signing request (.csr file) from Worldpay. This is linked to the merchant id

1. Register Merchant ID at <https://developer.apple.com/account/resources/identifiers/add/merchant>
 - a. The description field is for internal use. Hint: use sandbox and production so you can make distinction between test and production Apple Pay merchants
 - b. The identifier field should be the Merchant ID you've received from Worldpay
2. Create the certificates for the merchant at <https://developer.apple.com/account/resources/certificates/list>
 - a. Select 'development' from the All Types menu if you are registering your certificates for the sandbox
 - b. Click the + to register certificates
 - c. Create a 'Apple Pay Merchant Identity Certificate'
 - d. Select your Merchant ID
 - e. Upload the Certificate Signing Request you've received from Worldpay
 - f. Download the certificate file. Keep this somewhere safe
3. Domain validation:
 - a. You will need to host a text file on your test / production environment. This needs to be open to public or at least to Apple - https://developer.apple.com/documentation/apple_pay_on_the_web/setting_up_your_server
 - b. Edit your Merchant ID
 - c. Add your frontend domain(s) and you will receive a text file.
 - d. Add this to your webserver at the right location (written on the page)
 - e. Note that this has to be with a valid SSL and return text/plain
 - f. Press the verify button
4. Create certificates
 - a. Create a public and private key - `openssl req -out apple.csr -new -newkey rsa:2048 -nodes -keyout sandbox.key`
 - i. If the previous step throw a error like "CSR algorithm/size incorrect. Expected: RSA(2048)" run both commands

1. openssl genrsa -out sandbox.key 2048
2. openssl req -new -key sandbox.key -out apple.csr
- b. Upload the file apple.csr to the Apple Pay Merchant Identity Certificate
- c. Download the merchant_id.cer file you receive from Apple
- d. Create a pem file based on this certificate openssl x509 -inform der -in merchant_id.cer -out sandbox.pem
- e. Create a p12 file to be used in the Java backend openssl pkcs12 -export -in sandbox.pem -inkey sandbox.key -out apple-pay.p12 -name "Worldpay <your name> Apple Pay keystore"
- i. Secure your p12 file with a password!
5. Save the apple-pay.p12 in ext-worldpay/worldpayapi/resources/applepaycert
6. Update your worldpayApplePayHttpClient to have the right password and p12 file.

SAP Commerce 2105 Connector R4.0 - OCC Technical Implementation Guide

- [Worldpay OCC extension](#)
 - [Related Documentation](#)
 - [Endpoints on the WorldpayCartsController](#)
 - [Endpoints on the WorldpayOrdersController](#)
 - [Endpoints on WorldpayOccApi](#)
 - [Endpoints on the GooglePay](#)
 - [Endpoints on the ApplePay](#)
 - [Payment flows](#)
 - [Related Documentation](#)
 - [Payments flow without 3D secure](#)
 - [Payments flow with 3D secure](#)
 - [Payments flow with 3DS2](#)
- [Data adjustment on Worldpay communication](#)
 - [Handling session id in a stateless environment](#)
 - [Passing client IP address through OCC to Worldpay](#)
- [Test extension - worldpayextocctests](#)
 - [Test Data](#)
 - [The worldpayextocctests template extension only contains the basic OAuth 2 client configuration to get connected to the OCC API. The essential data is listed below.](#)
 - [Client-side encryption \(CSE\) test](#)
 - [Geb - browser automation tool](#)
 - [3D secure test](#)
- [Spartacus setup with 3DS Flex](#)

Worldpay OCC extension

The worldpayextoccc facilitates Worldpay payment on the shopping cart, placing orders with Worldpay payments, and handling the 3D secure authentication protocol when placing such an order.

It supports this using Worldpay's client-side encryption (CSE) in a B2C context.

The worldpayextoccc is an OCC extension that depends on the worldpayapi extension for Worldpay payments operations.

This functionality is supplied by the three controllers **WorldpayCartsController**, **WorldpayOrdersController**, **ApplePayController**.

All REST endpoints supplied by this extension supports URL encoded parameters and a body payload of either XML or JSON.

For documentation on the full hybrid OCC interface see:



Related Documentation

- [OCC API documentation, version v2](#)

Endpoints on the WorldpayCartsController

Method	Path	Parameters
POST	/users/{userId}/carts/{cartId}/worldpaypaymentdetails	The hybrid OOB PaymentDetails is extended with a cseToken
POST	/users/{userId}/carts/{cartId}/place-order	3D Secure - placing an order

Endpoints on the WorldpayOrdersController

Method	Path	Parameters
POST	/users/{userId}/worldpayorders	<ul style="list-style-type: none"> • cartId - the id of the used shopping cart • securityCode - the security code for the used credit card
POST	/users/{userId}/worldpayorders/3dresponse	<ul style="list-style-type: none"> • cartId - the id of the used shopping cart • paRes - the 3D protocols payer authentication response • merchantData - the merchant data used in the 3D protocol, this contains the Worldpay order code
POST	/users/{userId}/initial-payment-request	<ul style="list-style-type: none"> • cartId - the id of the used shopping cart • challengeWindowSize - size of the modal used for 3ds challenge (if required) <ul style="list-style-type: none"> • supported: 250x400, 390x400, 500x600, 600x400 • dReferenceId - reference id given during DDT • securityCode - CVC of the payment info
POST	/users/{userId}/worldpayorders/3dresponse	<ul style="list-style-type: none"> • cartId - the id of the used shopping cart • paRes - the 3D protocols payer authentication response • merchantData: Worldpay order code

Endpoints on WorldpayOccApi

Meth od	Path	Parameters
GET	/{{baseSiteId}}/worldpayapi/cse-public-key	n/a
GET	/{{baseSiteId}}/worldpayapi/ddc-3ds-jwt	n/a
POST	/{{baseSiteId}}/worldpayapi/challenge/submit	<ul style="list-style-type: none"> • TransactionId - transaction id sent by 3ds challenge (optional) • Response - arbitrary string (optional) • MD - worldpay order code

Endpoints on the GooglePay

Meth od	Path	Parameters	Reference links
POST	/checkout/multi/worldpay/googlepay/authorise-order	token <ul style="list-style-type: none"> - protocolVersion - The protocol Version - signature - The signature - signedMessage - The signed message billingAddress <ul style="list-style-type: none"> - address1 - The first line of the address - address2 - The second line of the address - address3 - The third line of the address - administrativeArea - The administrative area - countryCode - The country Code - locality - The locality of the address - name - The name of the receiver - postalCode - The postal code 	https://developers.google.com/pay/api/web/guides/resources/payment-data-cryptography

		- sortingCode - The shorting code	
POST	/users/{userId}/carts/{cartId}/googlepay-details		

Endpoints on the ApplePay


Method	Path	Parameters	Reference links
POST	/checkout/multi/worldpay/applepay/request-session	validationURL	
POST	/checkout/multi/worldpay/applepay/authorise-order	<p>Token</p> <ul style="list-style-type: none"> - ApplePayPaymentMethod <ul style="list-style-type: none"> -- displayName -- network -- type -- paymentPass <ul style="list-style-type: none"> --- primaryAccountIdentifier --- primaryAccountNumberSuffix --- deviceAccountIdentifier --- deviceAccountNumberSuffix --- activationState - transactionIdentifier - paymentData <ul style="list-style-type: none"> -- header <ul style="list-style-type: none"> --- ephemeralPublicKey --- publicKeyHash --- transactionId -- signature -- version -- data - billingContact <ul style="list-style-type: none"> -- phoneNumber -- emailAddress -- givenName -- familyName -- phoneticGivenName -- phoneticFamilyName -- addressLines[] -- subLocality -- locality -- postalCode -- subAdministrativeArea -- administrativeArea -- country -- countryCode - shippingContact: Same fields as billingContact 	https://developer.apple.com/documentation/apple_pay_on_the_web/applepaypaymentrequest

POST	/checkout/multi/worldpay/applepay/update-payment-method	paymentMethod	
GET	/({baseSiteId})/users/{userId}/carts/{cartId}/apple/payment-request	n/a	
POST	/({baseSiteId})/users/{userId}/carts/{cartId}/apple/request-session	<ul style="list-style-type: none"> validationURL: Apple API endpoint for requiring payment session 	
POST	/({baseSiteId})/users/{userId}/carts/{cartId}/apple/authorise-order	<ul style="list-style-type: none"> Token <ul style="list-style-type: none"> ApplePayPaymentMethod <ul style="list-style-type: none"> -- displayName -- network -- type -- paymentPass <ul style="list-style-type: none"> --- primaryAccountIdentifier --- primaryAccountNumberSuffix --- deviceAccountIdentifier --- deviceAccountNumberSuffix --- activationState transactionIdentifier paymentData <ul style="list-style-type: none"> -- header <ul style="list-style-type: none"> --- ephemeralPublicKey --- publicKeyHash --- transactionId -- signature -- version -- data billingContact <ul style="list-style-type: none"> -- phoneNumber -- emailAddress -- givenName -- familyName -- phoneticGivenName -- phoneticFamilyName -- addressLines[] -- subLocality -- locality -- postalCode -- subAdministrativeArea -- administrativeArea -- country -- countryCode shippingContact: Same fields as billingContact 	

Payment flows

The section gives an overview of the Worldpay payment flows.

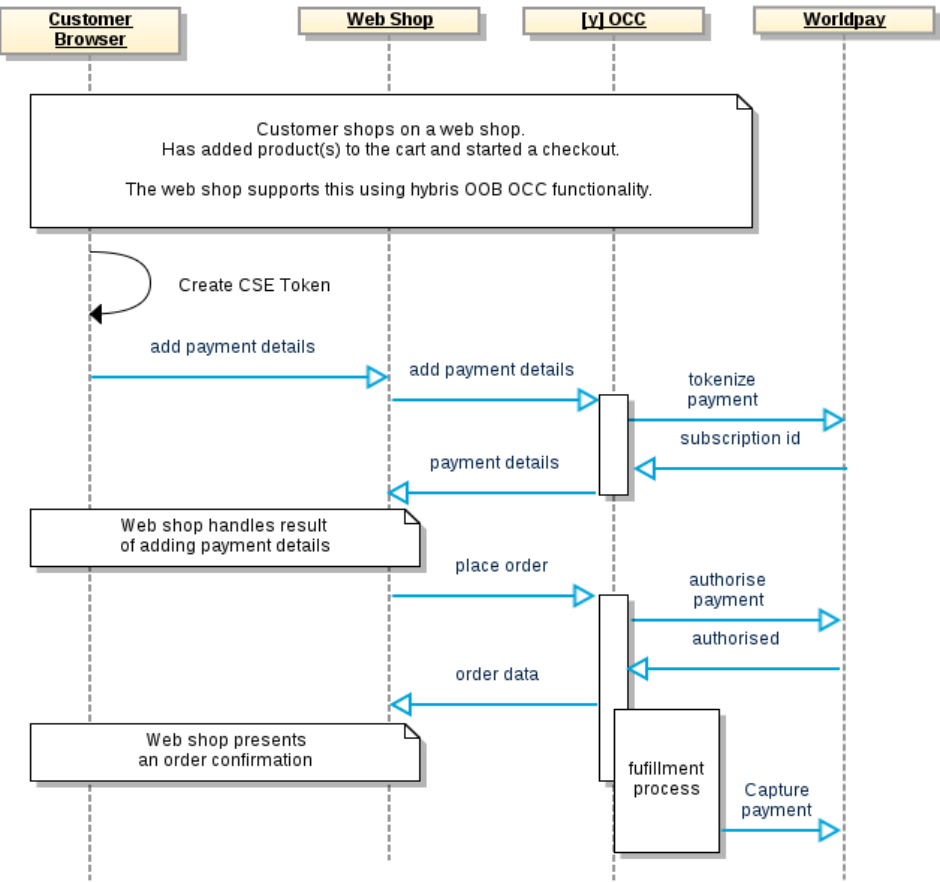
For a good walk-through of the customer buying process using OCC see:

 Related Documentation

- Customer Buying Process Scenarios

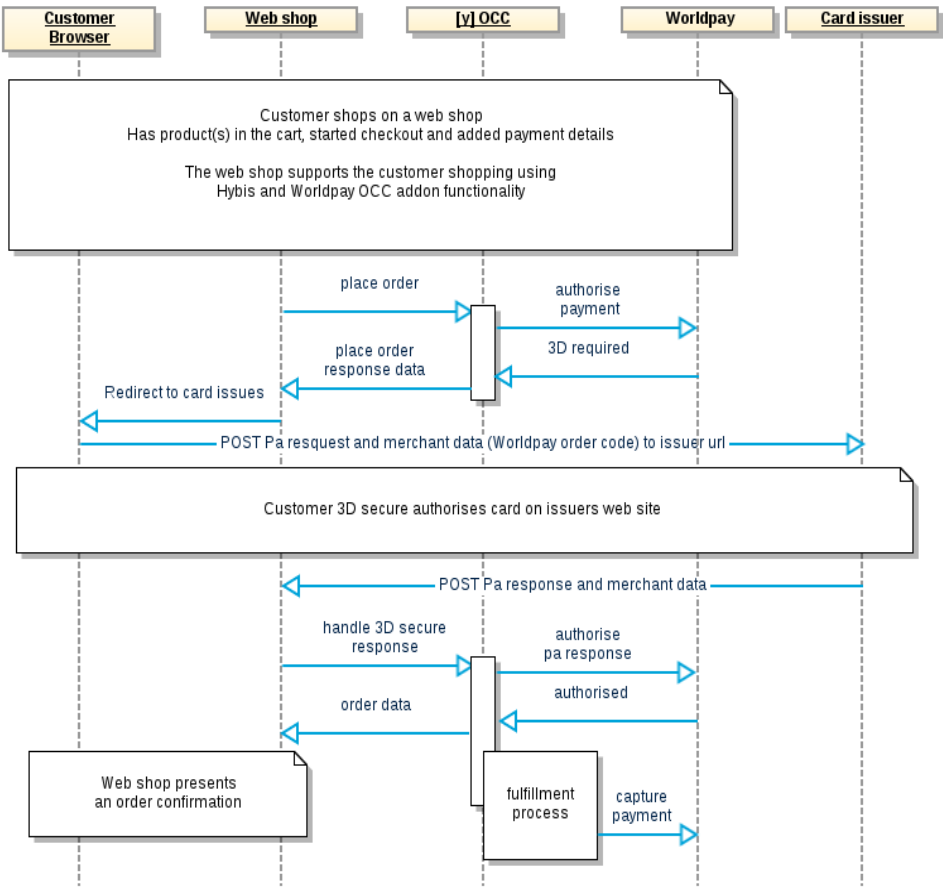
Payments flow without 3D secure

Worldpay OCC Payment Sequence Diagram



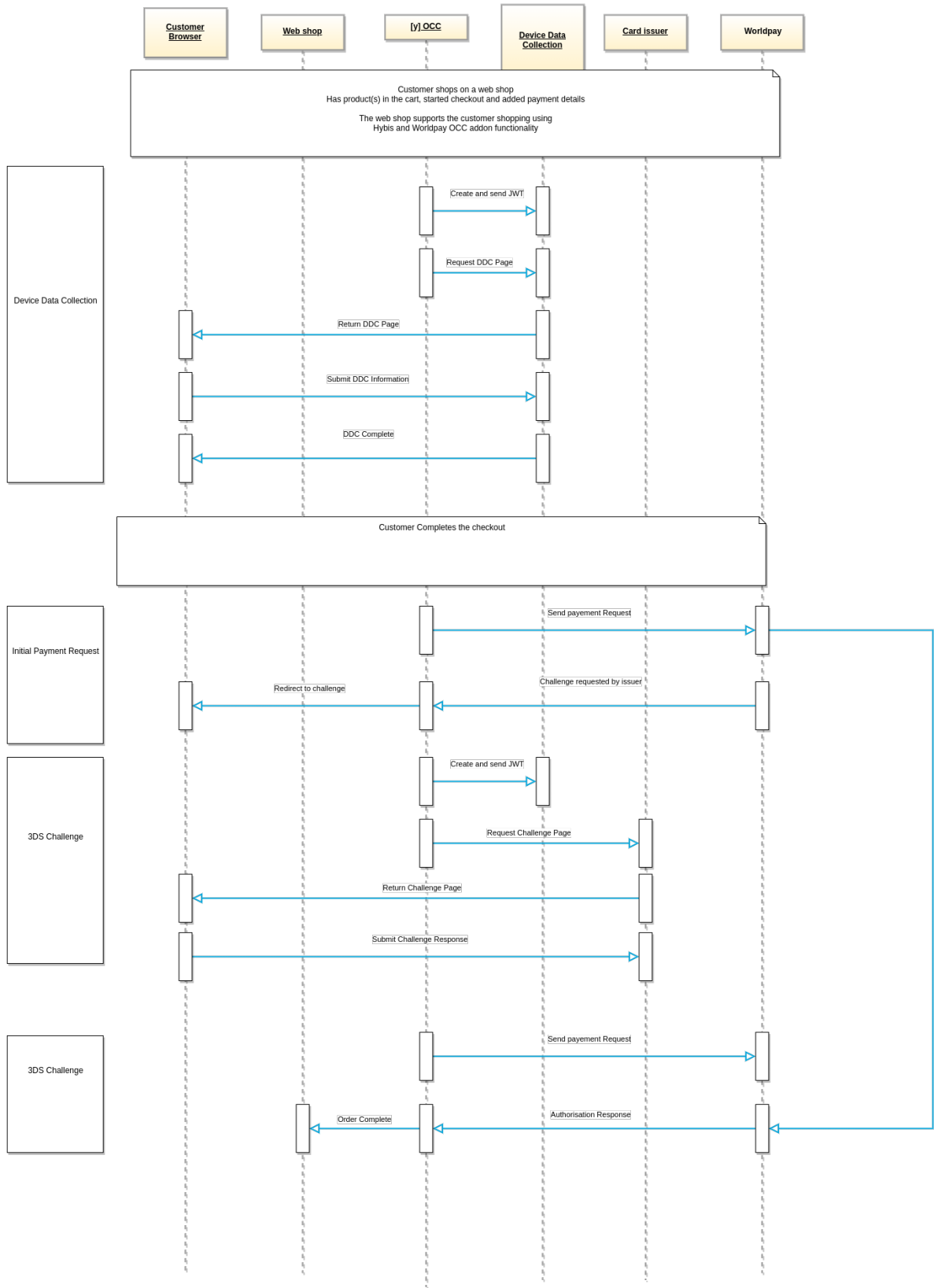
Payments flow with 3D secure

Worldpay OCC Payment with 3D Sequence Diagram



Payments flow with 3DS2

Worldpay OCC Payment with 3DS Flex Sequence Diagram



Data adjustment on Worldpay communication

The section covers some special cases of data management in the hybris Worldpay communication

Handling session id in a stateless environment

Worldpay requires that identical session ids are supplied when an order is supplied more than once.

In the 3D secure scenario, the order is submitted once on authorise and once when the pa response is validated in the second call.

OCC is a REST API that is stateless by design, so in this context, there is no session id on the request.

To solve this issue we hash the OAuth 2 token and apply this as a session id, hence 3D secure authentication has to be handled within the same OCC login session.

Passing client IP address through OCC to Worldpay

In order to secure that the correct customer IP address is passed to Worldpay the webshop implementer is responsible for parsing it through to OCC in an HTTP header property.

This is handled by a strategy, where the used header property name can be configured. The spring definition of this strategy is listed below. It contains a list of alternative header properties that can be used.

If this strategy is not used, the OCC AddOn would only have the calling web shops IP address to pass to Worldpay.

Due to load balancers and similar components, the strategy is also used in an accelerator storefront context.

worldpayapi-spring.xml

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

  <!-- Possible headers that contain the customer IP
  <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>
```

Test extension - worldpayextocctests

The Worldpay OCC AddOn's endpoints are tested using the Spock test framework as supplied with the Hybris template extension **yocctests**.

The tests are released together with the Worldpay OCC AddOn in the extension **worldpayextocctests**.

To execute all the tests in **worldpayextocctests** execute the following ant command:

```
ant all integrationtests -Dfailbuildonerror=yes -Dtestclasses.packages=com.worldpay.
worldpayextocctests.test.groovy.webservicetests.v2.spock.AllSpockTests
```

Test Data

The **worldpayextocctests** template extension only contains the basic OAuth 2 client configuration to get connected to the OCC API. The essential data is listed below.

essentialdataOAuthClientDetails.impex

```
INSERT_UPDATE OAuthClientDetails;clientId[unique=true];resourceIds;
scope;authorizedGrantTypes;authorities;clientSecret;
```

```

registeredRedirectUri
;client-side;hybris;basic;implicit,client_credentials;ROLE_CLIENT;
secret;http://localhost:9001/authorizationserver
/oauth2_implicit_callback;
;mobile_android;hybris;basic;authorization_code,refresh_token,password,
client_credentials;ROLE_CLIENT;secret;http://localhost:9001
/authorizationserver/oauth2_callback;
;trusted_client;hybris;extended;authorization_code,refresh_token,
password,client_credentials;ROLE_TRUSTED_CLIENT;secret; ;

```

Client-side encryption (CSE) test

In order to be able to test CSE in an OCC context where payment details are added to cart, we have to simulate a running browser where the Worldpay CSE javascript is executed.

The utility method below uses Geb to simulate the browser.



Geb - browser automation tool

- <http://www.gebish.org/manual/current/>

The browser accesses the cseTest.html page below, whose only task it is to load the Worldpay CSE javascript and supply a javascript function (generateCseToken) to execute the card encryption function.

When the page is loaded, generateCseToken is called and the resulting CSE token is obtained and passed back to the calling Spock test.

AbstractWorldpaySpockTest.groovy

```

protected getCseToken(cvc, cardHolderName, cardNumber, expiryMonth,
expiryYear) {
    def cseToken
    def browser = new Browser(driver: new FirefoxDriver())
    browser.go "file://" + (String) config.HTML_PATH + "/cseTest.html"
    cseToken = browser.js.generateCseToken
("1#10001#c745fe13416ffc5f9283f4 ...",
    cvc,
    cardHolderName,
    cardNumber,
    expiryMonth,
    expiryYear)
    browser.close()
    return cseToken
}

```

cseTest.html


```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>CSE Test Form</title>
  <script type="text/javascript" src="https://ajax.googleapis.com/ajax
/libs/jquery/1.12.2/jquery.min.js"></script>
  <script type="text/javascript" src="https://payments.worldpay.com
/resources/cse/js/worldpay-cse-1.0.1.min.js"></script>
  <script type="text/javascript" >
    function generateCseToken(publicKey, cvc, cardHolderName,
cardNumber, expMonth, expYear) {
      Worldpay.setPublicKey(publicKey);

      var data = {
        cvc: cvc,
        cardHolderName: cardHolderName,
        cardNumber: cardNumber,
        expiryMonth: expMonth,
        expiryYear: expYear
      };
      var encryptedData = Worldpay.encrypt(data, this.errorHandler);

      return encryptedData;
    }

    function errorHandler(errorCodes) {
      for (var index in errorCodes) {
        var errorCode = errorCodes[index].toString();
        alert(errorCode);
      }
    }
  </script>
</head>
<body></body>
</html>

```

3D secure test

In order to test the 3D secure flow, you need to be able to simulate the following steps after a place order has been called on the `WorldpayOrdersController`.

1. Redirect the customer's browser to the card issuer (in our case the Worldpay 3D secure simulator) supplying the pa request, the merchantData, and a returning term URL.
Again Geb and an HTML page are used to simulate this. The HTML page auto submits the supplied data in the form to the Worldpay 3D secure simulator.
2. Now the browser is located on the Worldpay 3D secure simulator, where the utility function chooses the outcome of the simulation and clicks the simulators button to proceed.
3. The simulator posts to the term URL. This hits the **Worldpay3DResponseMockController** method shown below, which returns a page where the pa response can be obtained and passed back to the calling Spock test.

The below test method illustrates how the three steps have been implemented.

AbstractWorldpaySpockTest.groovy

```
protected handleThreeDSecureInBrowser(issuerUrl, paRequest,
merchantData, authorisationResponse) {

    def browser = new Browser(driver: new FirefoxDriver())

    def termUrl = getDefaultHttpsUri() + "/worldpayresponsemock
/3dresponse"
    def autoSubmitUrl = "file://" + (String) config.HTML_PATH + "
/threeDSecureTest.html?" +
        "IssuerUrl=" + URLEncoder.encode(issuerUrl, "UTF-8") +
        "&PaReq=" + URLEncoder.encode(paRequest, "UTF-8") +
        "&MD=" + URLEncoder.encode(merchantData, "UTF-8") +
        "&TermUrl=" + URLEncoder.encode(termUrl, "UTF-8")

    browser.go autoSubmitUrl

    // The threeDSecureTest.html page auto submits and forwards to the
    // worldpay 3D simulator page (the issuer url)
    browser.$("form").paResMagicValues = authorisationResponse

    // On the worldpay 3D simulator we select the given
    authorisationResponse and click the submit button
    browser.getPage().$(org.openqa.selenium.By.className("lefty")).
    click()

    // We are now on a mock endpoint in the worldpayresponsemock
    extension which collects the Pa response
    def paRes = browser.getPage().$(org.openqa.selenium.By.className
("PaRes")).value()
    browser.close()

    return paRes
}
```

Worldpay3DResponseMockController.java

```
@RequestMapping (method = POST)
public String mockWorldpayResponse(final ModelMap model, final
```

```

HttpServletRequest request) {

    String paRes = request.getParameter("PaRes");
    String merchantData = request.getParameter("MD");

    model.put("paRes", paRes);
    model.put("merchantData", merchantData);

    return "pages/threeDSecureResponse";
}

```

Spartacus setup with 3DS Flex

To make Spartacus work with 3DS Flex it's necessary to set up correctly the property:

```

xss.filter.header.Content-Security-Policy=frame-ancestors 'self'
http\://electronics.localhost.com\:4200 https\://electronics.localhost.
com\:4200

```

This property will let hybris embed an iframe from Spartacus web app.

SAP Commerce 2105 Connector R4.0 - Technical Implementation Guide

- [Reader's guide](#)
 - [High-Level Architecture](#)
 - [Hybris Commerce Suite Extensions](#)
 - [Connector](#)
 - [worldpayapi](#)
 - [Front end](#)
 - [worldpayaddon](#)
 - [worldpayb2baddon](#)
 - [worldpayaddoncommons](#)
 - [worldpayextocc](#)
 - [worldpaysampledatabaddon](#)
 - [worldpayfulfilment](#)
 - [worldpayoms](#)
 - [worldpayaddonbackoffice](#)
 - [worldpaynotifications](#)
 - [Testing](#)
 - [worldpayocceexttests](#)
 - [worldpayresponsemock](#)
 - [Order notification mock](#)
 - [AddOn Integration with Hybris Payment Module](#)
 - [New Accelerator Checkout Flow](#)
 - [Order Cancellation](#)
 - [Extended Types](#)
 - [Update Worldpay's DTD](#)
 - [Supported Environments](#)
 - [Limitations](#)
 - [Languages](#)

Reader's guide

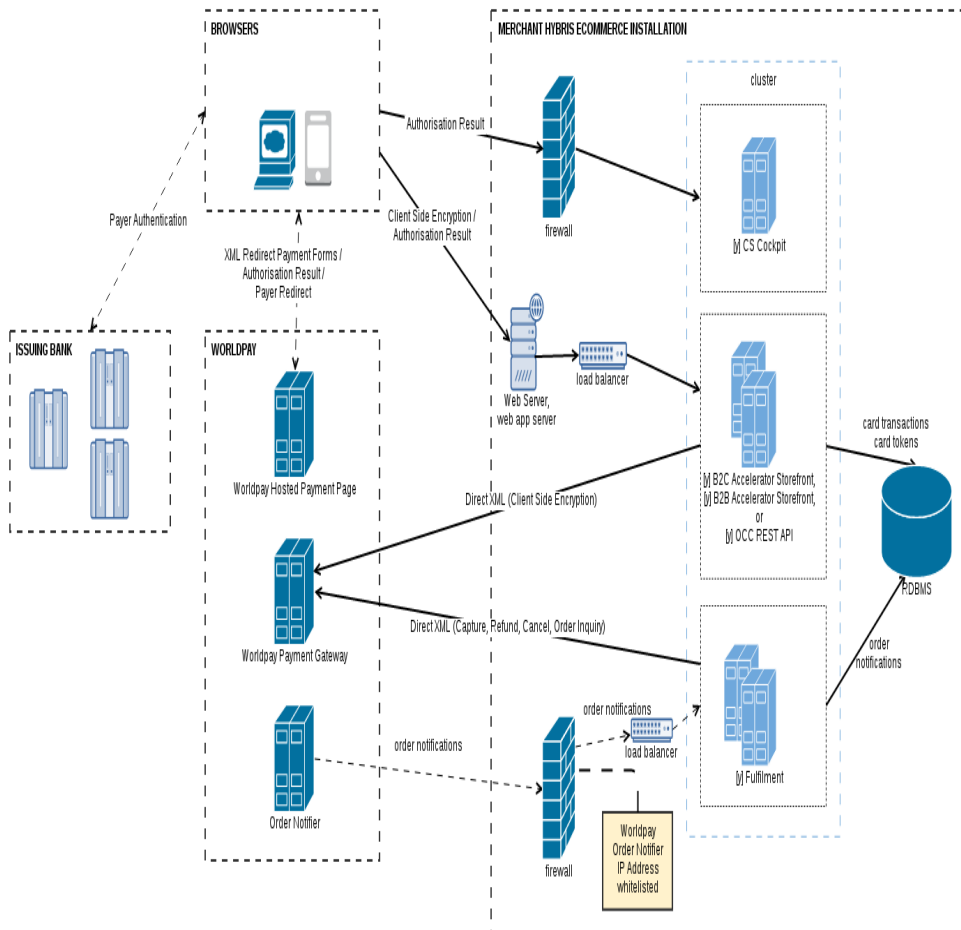
This technical implementation guide will cover the use of Worldpay as a payment provider mainly in a B2C context. Some sections will apply to both B2C and B2B that only apply to either B2C or B2B will be marked as **B2C only** or **B2B only**

Technical Implementation Guide

All Worldpay Payment Commands are executed asynchronously and therefore will respond initially in a *pending* state. Once the transaction has completed, Worldpay pushes notifications via the Order Notifier back to a configured endpoint on a Hybris Commerce Suite node. The endpoint URL is configured in the merchant account in the Worldpay Merchant Account Interface Tool. The endpoint (exposed by installing the **worldpaynotifications** extension) serialises these notifications as **WorldpayOrderModification** records into the Hybris Commerce Suite database.

High-Level Architecture

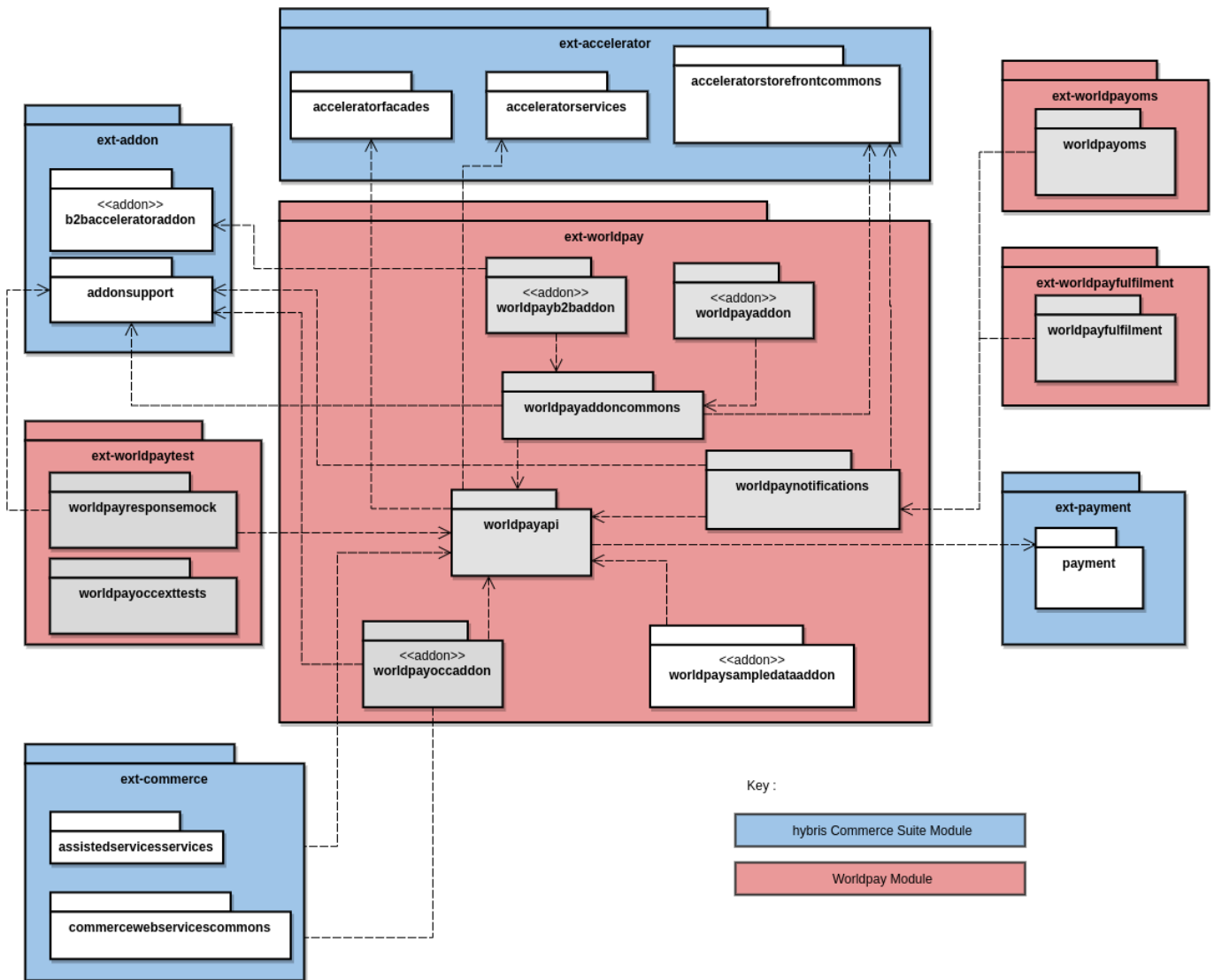
The following diagram outlines at a high level, the system components (Worldpay, Issuing Bank and Hybris) together with the high-level interactions that fulfil the functionality delivered by the Worldly AddOn.



Hybris Commerce Suite Extensions

The Worldpay Hybris AddOn is shipped in 4 separate modules **ext-worldpay**, **ext-worldpayfulfilment**, **ext-worldpayoms**, **ext-worldpaytest**.

The Worldpay AddOn extends the Hybris Commerce Suite providing integration with Worldpay Hosted Order Page, Payment API, Notification, Fraud, CSE and tokenisation functionalities.



Connector

worldpayapi

Worldpay API Gateway Plugin point for the Hybris Commerce Suite Payment Service.



Related Documentation

- [Payment Extension](#)

Front end

worldpayaddon

AddOn for the Hybris B2C Accelerator. This adds a new Checkout Flow that is optimised for Worldpay HOP (Hosted Order Page), CSE/Direct both supporting tokenisation functionality.



Related Documentation

- [Configurable Checkout](#)
- [AddOn Concept](#)

worldpayb2baddon

AddOn for the Hybris B2B Accelerator. This adds a new Checkout Flow that is optimised for Worldpay CSE/Direct supporting tokenisation functionality.



Related Documentation

- [Configurable Checkout](#)
- [AddOn Concept](#)
- [B2B Checkout and Order Process](#)

worldpayaddoncommons

Commons extension for B2C and B2B AddOn's.

worldpayextocc

Extension for the Hybris OCC REST API. This adds payment details and place order in a B2C context. Support CSE and 3D secure.



Related Documentation

- [OCC Architecture Overview](#)
- [OCC extension](#)

worldpaysampledatabaddon

AddOn that adds sample data on the electronics and apparel sites. Contains several APM configurations, adds some countries, currencies and delivery zones. Also adds CMS components to the payment and billing page for the electronics and apparel content catalogs enabling the different APMs.

The data provided here could serve as an example of how to import different APMs and how to configure the different components in the payment and billing page.

worldpayfulfilment

Adds Worldpay related functionalities and customisations to payment integrations for the Hybris Accelerator Order Fulfilment Process.



Related Documentation

- [yacceleratorfulfilmentprocess Extension](#)

worldpayoms

Adds Worldpay related functionalities and customisations to payment integrations for the Hybris Order Management System.



Related Documentation

- [Order Management for SAP Hybris Commerce](#)

worldpayaddonbackoffice

Extension that includes configuration files so items created in the AddOn are organised and visible in the backoffice Hybris application.



Related Documentation

- [Backoffice Framework](#)

Notification

worldpaynotifications

Extension providing an HTTP endpoint to receive and process Worldpay Order Notification messages.

Testing

worldpayocctesttests

Extension containing Spock test that verifies the endpoints implemented at the **worldpayextocc**. See our [OCC Documentation](#) for usage and details.

worldpayresponsemock

The connector contains an extension that offers two functionalities:

Direct/redirect responses mock

Acting as the Gateway itself and replying to Direct and Redirect requests. The mock replies with pre-fabricated happy-flow responses to authorize and capture requests.

To enable the mock, set the following properties:

worldpayapi/project.properties

```
# Valid values for environment are MOCK, TEST and PRODUCTION
worldpay.config.environment=MOCK
worldpay.config.endpoint.MOCK=http://<YOUR-SITE>:9001
/worldpayresponsemock/mock
```

You also need to add the property `-Djavax.xml.accessExternalDTD=all` to the property `tomcat.generaloptions` in your `local.properties`

Order notification mock

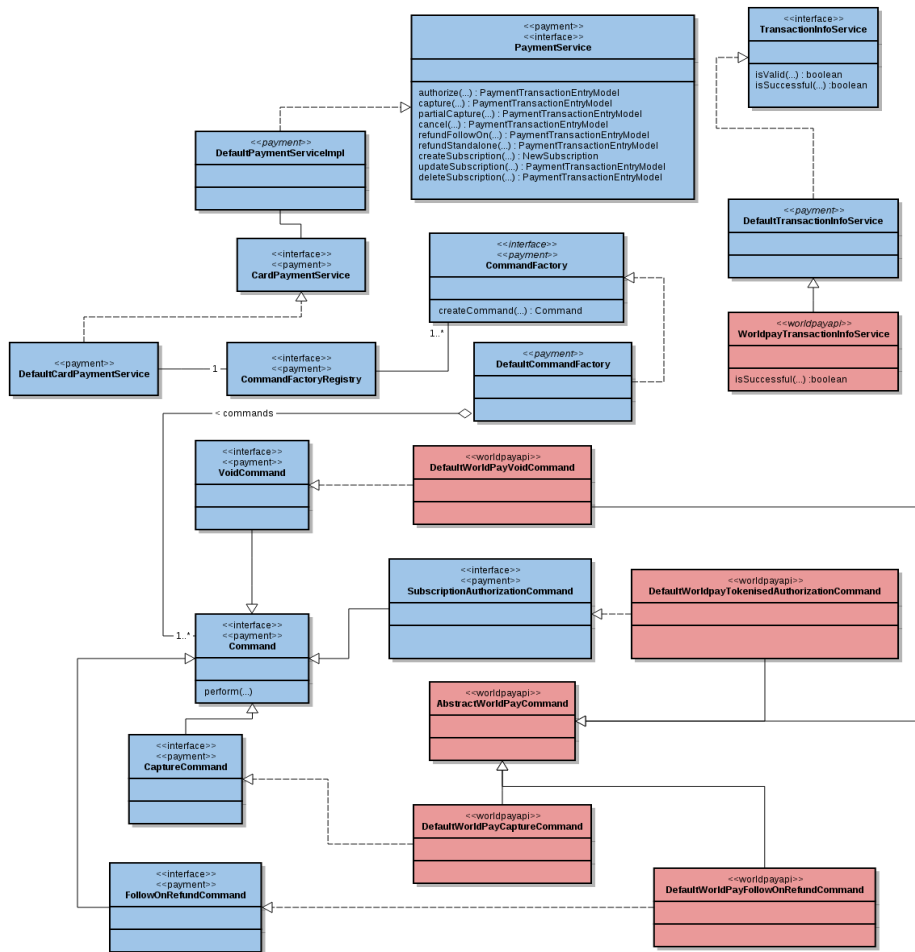
The Worldpay Order Notification System requires the Hybris Commerce Suite to expose an internet-facing Webservice endpoint. To facilitate development and test automation, the AddOn ships a test harness to mockup notification system responses in development environments. This is an optional installed Hybris extension with a simple Web front-end that can be used to simulate order notifications updates from Worldpay. Write in the Order Code (requestId, found in the backoffice or in the logs) and send the response you wish generated.



AddOn Integration with Hybris Payment Module

The Worldpay extension plugs directly into the Hybris Payment Module by providing Worldpay connectivity to authorization, capture, refund, void and subscription commands. The Worldpay implementations of these commands then use the Core Connector Library to communicate with the Worldpay Service Gateway.

The following class diagram shows the points in which Worldpay plugs into the Hybris Payment Service.

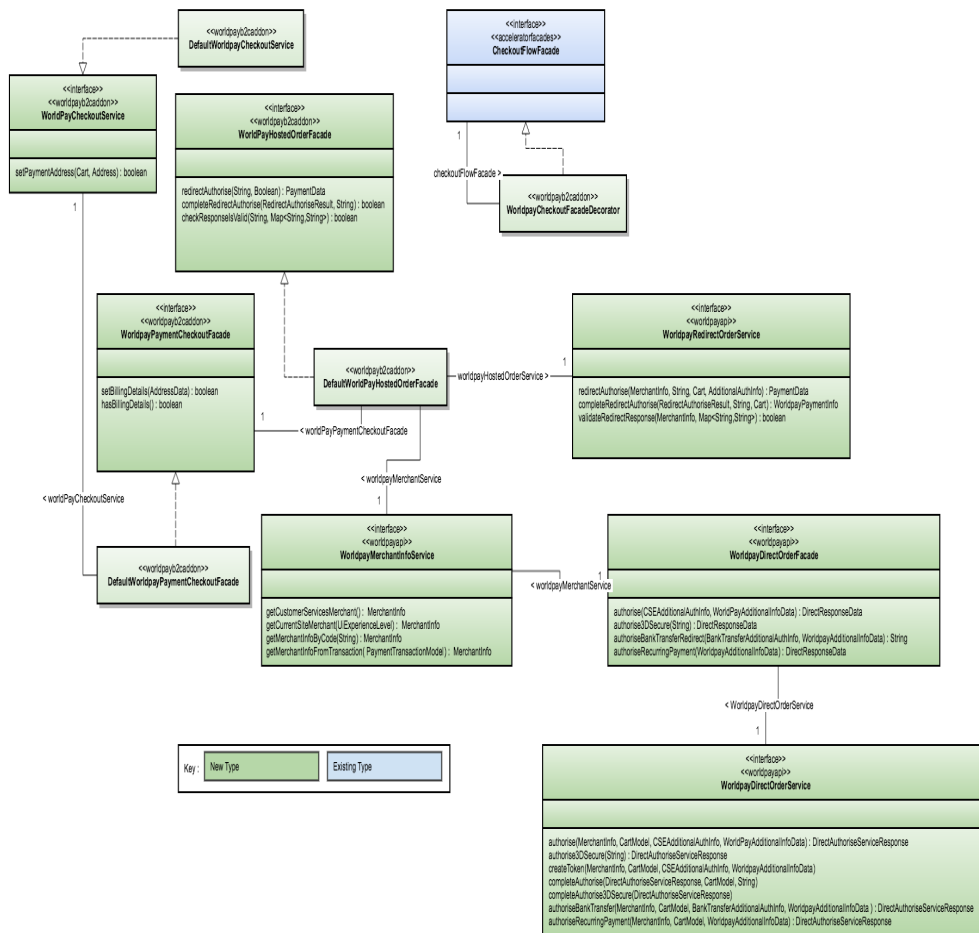


It is important to note, with the Worldpay integration, confirmation of the result of executing the command comes asynchronously via Worldpay posting to the Order Notification AddOn. Therefore Commands will return Payment Transaction Entries with a pending status.

New Accelerator Checkout Flow

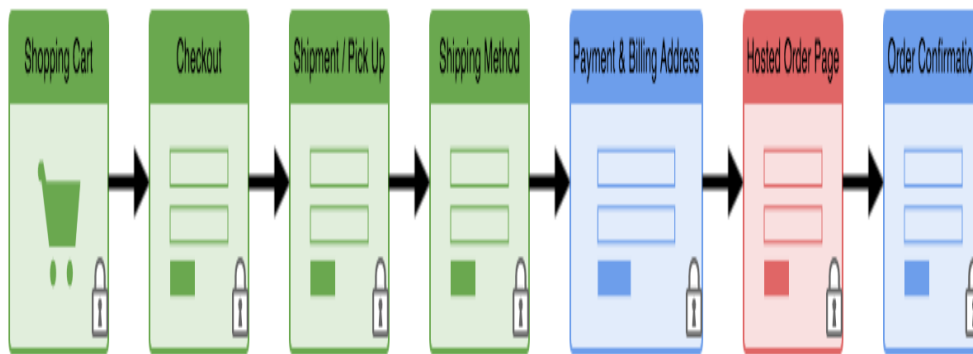
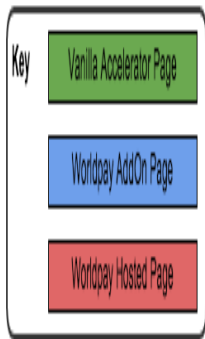
Facades & Services - B2C only

The Accelerator Service Layer and Facades have been extended to support functionality including XML Redirect Hosted Order Payment, Pay As Order and an XML Direct using Client-side encryption.



HOP - Redirect Flow - B2C only

The following diagram depicts the modifications made by the Worldpay B2C Accelerator AddOn which adds a Worldpay compatible Checkout Flow to replace the standard B2C Accelerator flow.



Payment & Billing Address Page - B2C only

This page has been customised to:

- Display a list of available payment methods (such as Cards and Alternative Payment Methods) which are CMS-driven.
- Display Terms & Conditions checkbox which is a required field in order to proceed the Hosted Order Page.
- JavaScript library added to handle the User Experience.

Order Summary Page - B2C only

This page has been removed from the vanilla Accelerator HOP flow. This is because the authorisation happens on the Worldpay Hosted Order Page and there is no need to ask the User to place the order. The order is placed at the end of the HOP page callback.

Order Confirmation Page - B2C only

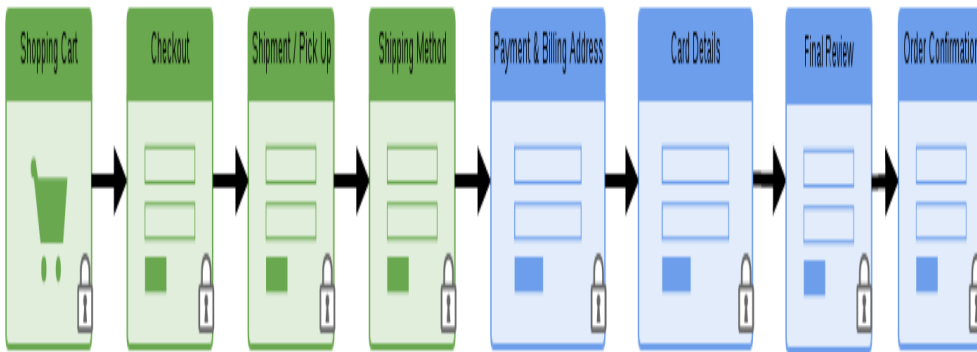
Small customisation to hide Payment Details in the event of the Notification not yet being received. Once the application has received the notification, it will show either the payment details about the credit/debit card used (obfuscated) or the Alternative Payment Method used to place the order.

Tokenisation - B2C only

When the customer chooses to save their card details in the Payment and Billing page, the request sent to Worldpay contains an instruction to tokenise the card. The corresponding authorisation order modification message will contain the token information.

Client-Side Encryption Flow

The following diagram shows how the Client-Side Encryption (CSE) payment flow has been implemented in the AddOn. This flow is only available for credit/debit card payments.



The only difference from the redirect-flow is that the user is no longer taken to the Hosted Order Page for filling in their card details. Card details are instead encrypted on the customer's browser and the encrypted payload is submitted to the Merchants Web storefront by a form post on the Card Details page. This encrypted data is then passed on to Worldpay using Direct XML integration.

Tokenisation

When the customer chooses to save their card details in the Card Details page, the direct request sent to Worldpay contains an instruction to tokenise the card. The direct response will contain the token information. The AddOn will store this information in the payment info associated with the order.

The response from Worldpay may contain a tokenEvent with the value "NEW" or "MATCH". NEW means that is a new card tokenised and stored by Worldpay, whereas MATCH means that the card has already been tokenised and the customer is not using an already saved card. In this case, the AddOn will look for an already tokenised and saved card associated to the customer and will attach the found one to the order and payment transaction. If there is no such payment info associated to the customer, a new one with the token information will be created.

Notifications

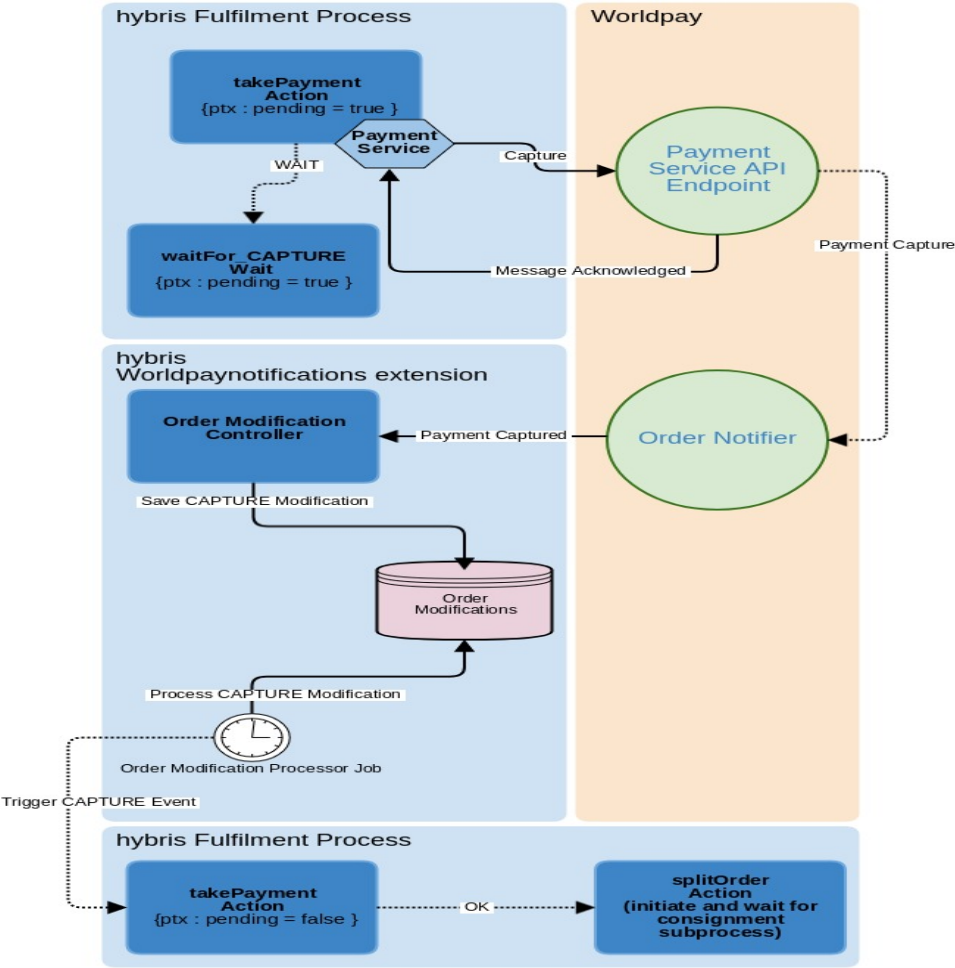
All Worldpay Payment Commands are executed asynchronously and therefore will respond initially in a *pending* state. Once the transaction has completed, Worldpay pushes notifications via the Order Notifier back to a configured endpoint on a Hybris Commerce Suite node. The endpoint URL is configured in the merchant account in the Worldpay Merchant Account Interface Tool. The Endpoint (exposed by installing the **worldpaynotifications** extension) serialises these notifications as **WorldpayOrderMofication** records into the Hybris Commerce Suite database.

Tokenisation

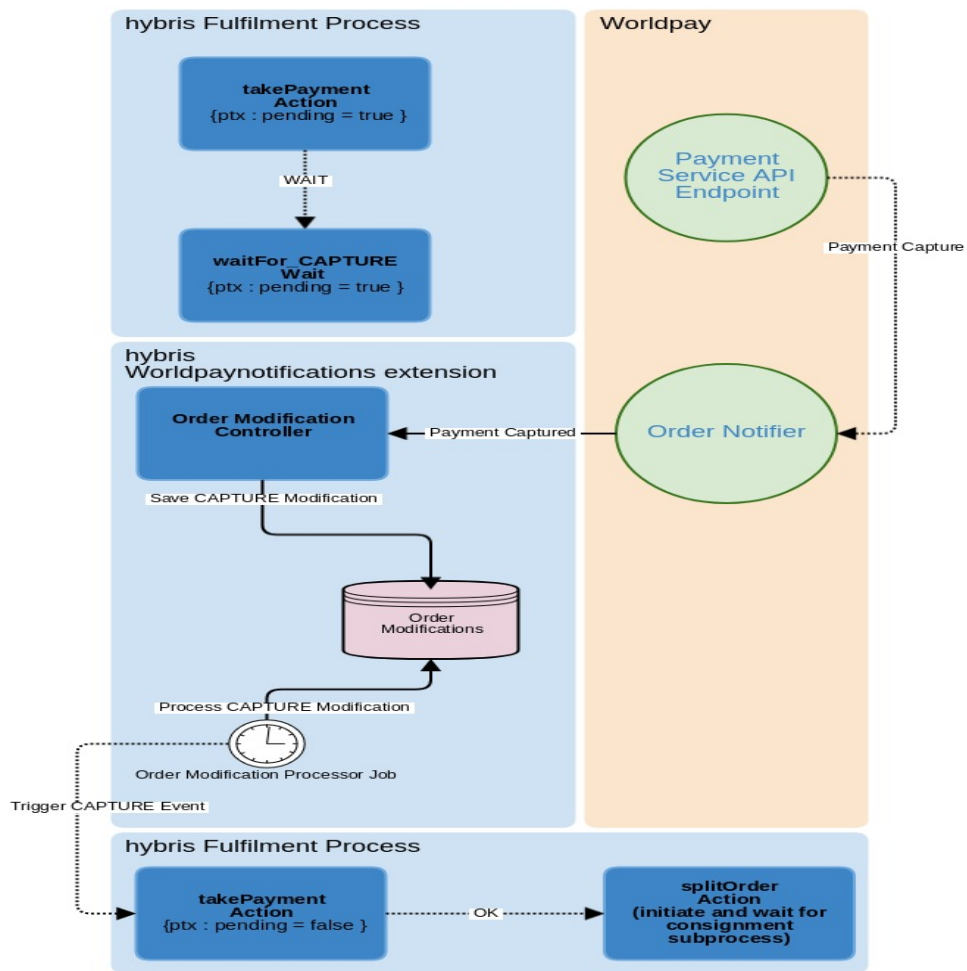
When the application receives the notification with the token information in the Redirect Flow, the AddOn will store this information in the payment info associated to the order.

The response from Worldpay may contain a tokenEvent with the value "NEW" or "MATCH". NEW means that is a new card tokenised and stored by Worldpay, whereas MATCH means that the card has already been tokenised and the customer is not using an already saved card. In this case, the AddOn will look for an already tokenised and saved card associated to the customer and will attach the found one to the order and payment transaction. If there is no such payment info associated to the customer, a new one with the token information will be created.

To illustrate the flow of a typical payment capture request for a **credit/debit card payment** use-case from Hybris Commerce Suite to Worldpay and then back into Hybris Commerce Suite, the following diagram is shown:

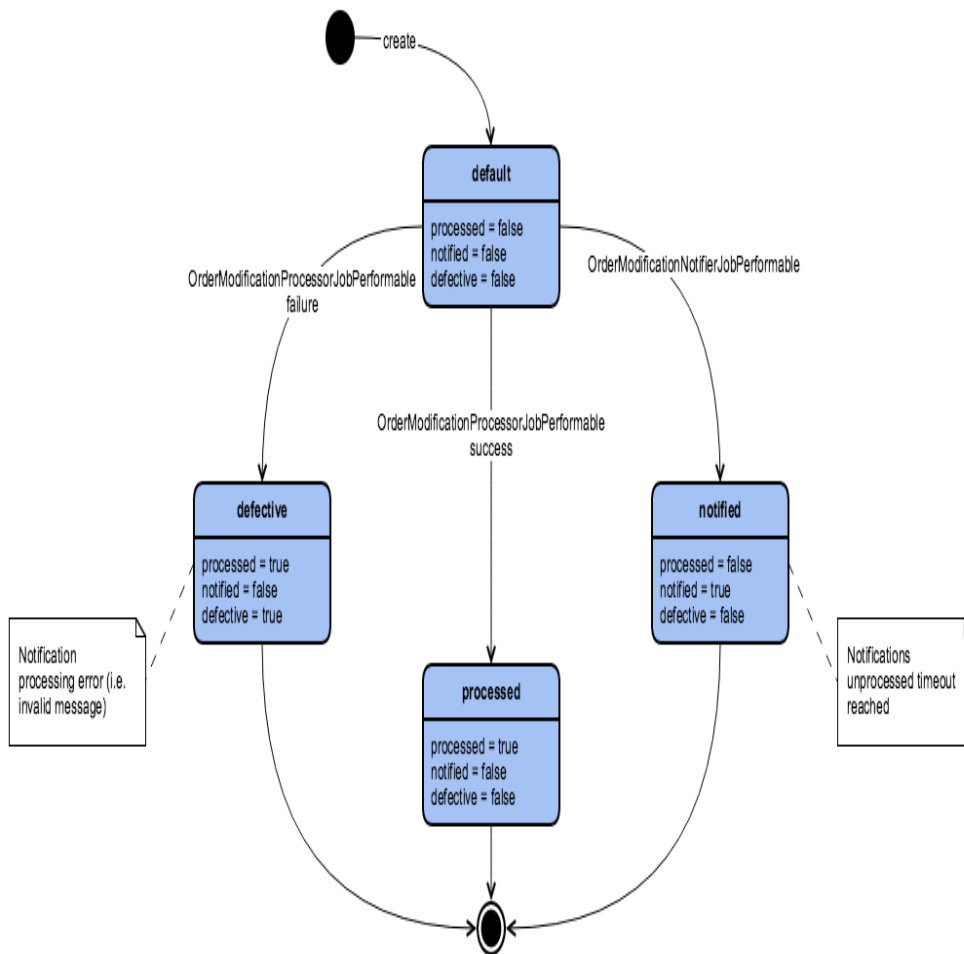


To illustrate the flow of a typical payment capture request for an **Alternative Payment Method (APM) payment** use-case from Hybris to Worldpay and then back into Hybris, the following diagram is shown:



Notification States

The following state diagram describes the states, transitions and possible attribute values of a **WorldpayOrderModification** record in the Hybris Commerce Suite database.



Notification Cleanup

WorldpayOrderModification's can build up over time, therefore two types of cleanup Cron Job's have been provided with the AddOn.

- Cleanup of processed notifications
 - A scheduled Cron Job will retrieve the Modification records which have been processed after a certain amount of time (configurable) and remove them from the system.
- Cleanup of unprocessed notifications
 - A scheduled Cron Job will retrieve Modification records which have been sitting unprocessed for a certain amount of time (configurable) and raise a Ticket for each of them.

Order Fulfilment Process

To help accelerator development of best practice payment fulfilment workflows, reference *Order Fulfilment Processes* have been supplied with the AddOn. Depending on whether you use Hybris Order Management System or the Accelerator Fulfilment Process you can add the **worldpayoms** or the **worldpayfulfilment** extension into your installation (see [Installation and Usage](#)).

When an order is placed in the Hybris Commerce Suite, it is sent to the process engine for execution by the order fulfilment process. There, the order is processed by a number of actions each having a different purpose. In **WorldpayCheckAuthorizeOrderPaymentAction** the amount from **PaymentTransactionEntries** - of type **AUTHORIZED** - will be compared to order total. If they don't match the order status will change to **CHECKED_INVALID**, and an **OrderHistoryEntry** will be added. This could happen if the cart has been modified during payment. For error handling, the **PaymentTransactions** can be found in Backoffice on the order Administration tab.



When comparing order total and authorised amount the difference is validated against a tolerance. The default tolerance is 0.01, but the tolerance level is configurable by property

```
worldpayapi.authoriseamount.validation.tolerance=0.01
```

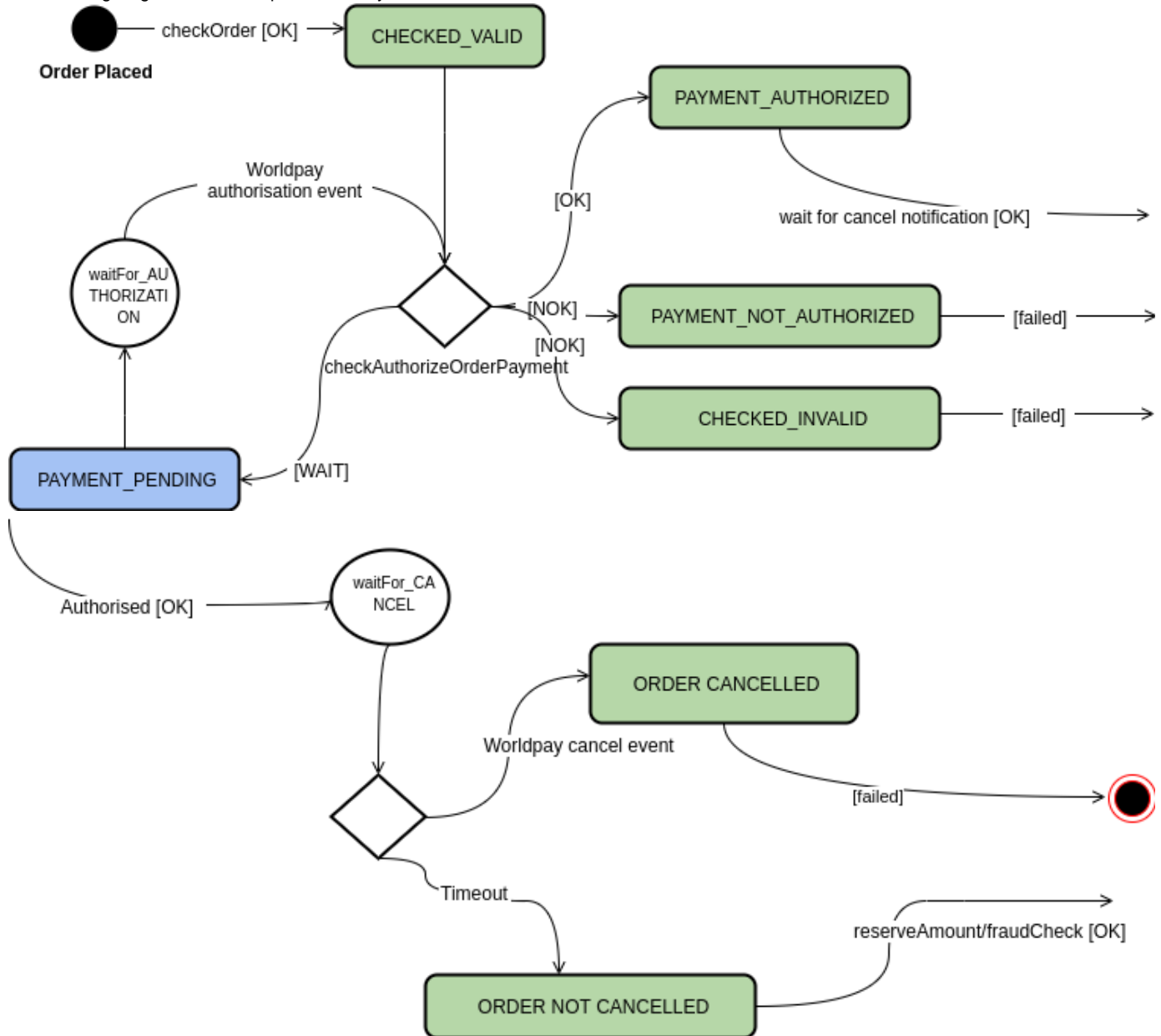
When all the actions have been executed, the order reaches its final state (be it error or success).

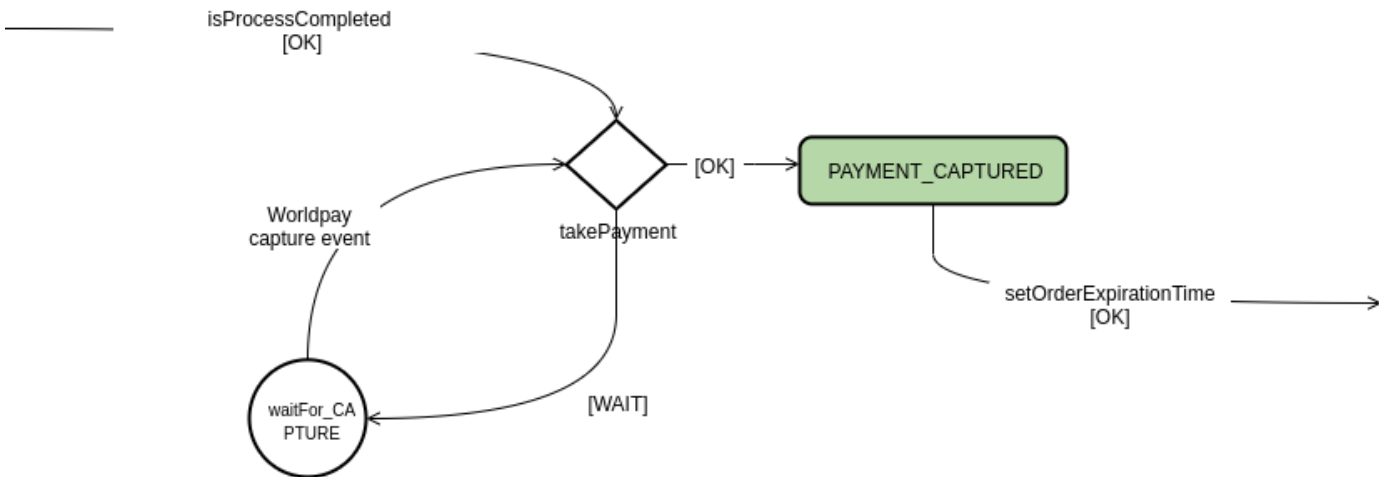
Given the asynchronous nature of Worldpay Order Modification messages, a number of WAIT states have been added to the reference process. Since every payment transaction entry is created in a pending state (exception for APM's), the WAIT will pause the fulfilment process until the confirmation of the said payment transaction has been confirmed (or denied). There is one WAIT for every Order Modification message we handle in the AddOn (AUTHORIZATION, CANCEL and CAPTURE). When the AUTHORIZATION message has been processed the *Order Fulfilment Process* waits for a possible cancel notification, but once it has reach the timeout for waiting for a cancel notification it continues until the **takePayment** step.

When an Order Modification message is received from Worldpay, the application persists the Modification message in the database and returns an acknowledgement (OK) response to the Order Notifier. A Cron Job (Order Modification Processor Job) is responsible to go through the unprocessed Order Modifications and process them. Different type of messages require different processing but for all of them, once processed, an event is triggered which the WAIT is waiting for. Once the WAIT receives the event, the Order Fulfilment process continues.

In the special case of APM, the order process will wait for the CAPTURE modification message, when it reaches the application, the transaction entry is created in a non-pending state. The order process will not issue a CAPTURE command to Worldpay when the payment was made through an APM except Klarna APM, which actually does a CAPTURE command.

The following diagrams show the parts of the Hybris Order Process where the WAIT states have been added:





Every order for which the authorisation notification has not been received from Worldpay will have the PAYMENT_PENDING status set and will stay in the waitFor_AUTHORIZATION wait state until such notification has been received. Depending on the outcome of the notification, the order processing will continue (authorisation successful) or will fail (authorisation not successful).

Return process

There is a customised return-process available if the chosen AddOn installation contains the OMS functionality. The modifications applied to this return-process adds a new waiting step for the REFUND notification.

worldpayoms/process/return-process.xml

```

    <wait id="waitForConfirmOrCancelReturnAction" prependProcessCode="
true" then="failed">
        <case event="ConfirmOrCancelRefundEvent">
            <choice id="cancelReturn" then="cancelReturnAction"/>
            <choice id="approveReturn" then="approveReturnAction"/>
        </case>
    </wait>

    <action id="cancelReturnAction" bean="cancelReturnAction">
        <transition name="OK" to="success"/>
    </action>

    <action id="approveReturnAction" bean="approveReturnAction">
        <transition name="OK" to="printReturnLabelAction"/>
    </action>

    <action id="printReturnLabelAction" bean="printReturnLabelAction">
        <transition name="OK" to="printPackingLabelAction"/>
    </action>

    <action id="printPackingLabelAction" bean="printPackingLabelAction">
        <transition name="OK" to="waitForGoodsAction"/>
    </action>

    <wait id="waitForGoodsAction" prependProcessCode="true" then="
failed">

```



```

        <case event="ApproveOrCancelGoodsEvent">
            <choice id="cancelReturn" then="cancelReturnAction"/>
            <choice id="acceptGoods" then="acceptGoodsAction"/>
        </case>
    </wait>

    <action id="acceptGoodsAction" bean="acceptGoodsAction">
        <transition name="OK" to="captureRefundAction"/>
    </action>

    <action id="captureRefundAction" bean="captureRefundAction">
        <transition name="OK" to="waitFor_REFUNDED"/>
        <transition name="NOK" to="waitForFailCaptureAction"/>
    </action>

    <!-- New WAIT to be kicked off when the transaction is Refunded by
worldpay -->
    <wait id="waitFor_REFUNDED" then="successCaptureAction"
prependProcessCode="false">
        <event>${process.code}_REFUND_FOLLOW_ON</event>
    </wait>

    <wait id="waitForFailCaptureAction" prependProcessCode="true" then="
failed">
        <case event="FailCaptureActionEvent">
            <choice id="bypassCapture" then="taxReverseAction"/>
            <choice id="cancelReturn" then="cancelReturnAction"/>
        </case>
    </wait>

```

The action captureRefundAction is also customised:

worldpayoms-spring.xml

```

    <alias name="worldpayCaptureRefundAction" alias="
captureRefundAction"/>
    <bean name="worldpayCaptureRefundAction" class="com.worldpay.
worldpayoms.fulfilmentprocess.actions.order.
WorldpayCaptureRefundAction" parent="abstractAction">
        <property name="paymentService" ref="paymentService"/>
        <property name="refundAmountCalculationService" ref="
refundAmountCalculationService"/>
    </bean>

```

Order Cancellation

The Worldpay AddOn enables the Call Centre Agents to cancel orders. Cancelling an order in Hybris sends a CANCEL request to Worldpay to reverse ringfenced funds on the customers payment method.

If the order was paid through an APM, or the payment method has not yet been communicated by the Worldpay Notification System, the order cannot be cancelled with the out of the box handling of the AddOn. If the Hybris installation is customised to handle multiple payments for the same order and one of these is an APM, the order will not be cancellable. This functionality can of course be changed on project once business rules have been realised.

In case of fraud review, the "Reject" action in the Backoffice will place the order in a WaitForCleanUp state. By default Hybris cancels the order through the cleanUpFraudOrderCronJob triggering the "VoidCommand" action, which is customised to send a CANCEL request to Worldpay.

Risk Guardian

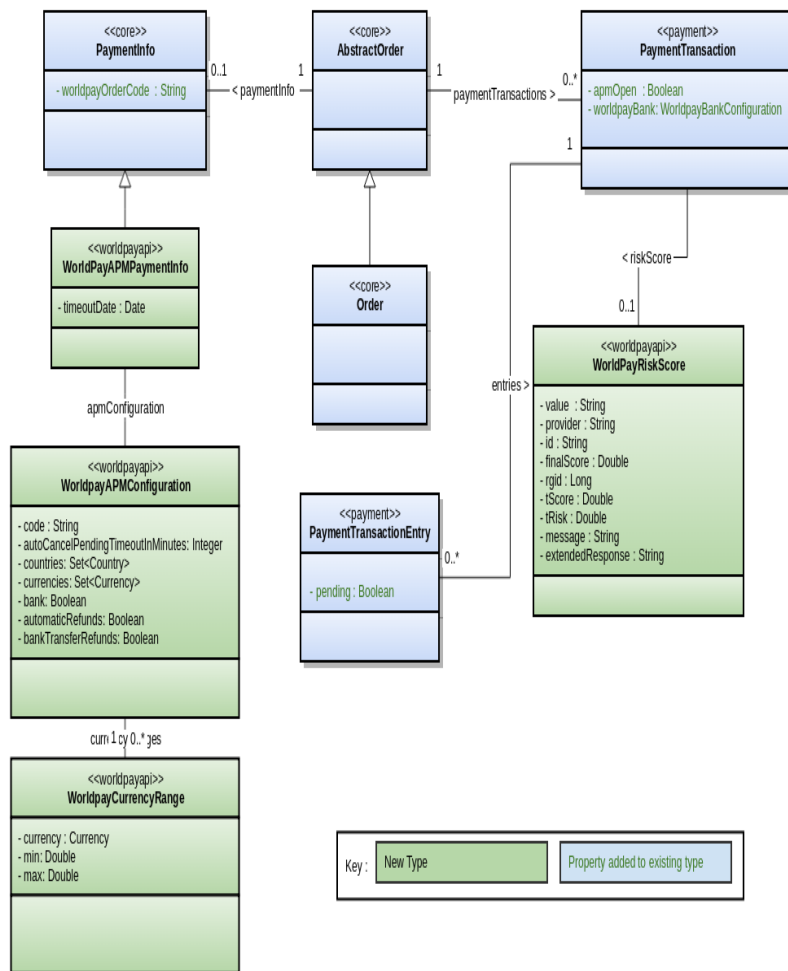
Risk Guardian is an advanced fraud detection tool that can be activated on a **Production** Worldpay Merchant Account. Extra fraud score information is returned with the Authorisation Response which is then captured against the Payment Transaction.

Out of the box the information is captured and displayed (in Customer Service tools) and orders with fraud scores greater than a configured threshold (**project.properties**) are held for review.

Extended Types

The Worldpay AddOn extends the Hybris Commerce Suite data model in a few area's (predominantly payment).

The main customisations are to introduce the concept of Pending Payment transactions, Fraud Report information and to support a more generic Payment type accommodating Worldpay's support for 200+ Alternative Payment Methods.



Update Worldpay's DTD

From time to time Worldpay releases a new version of their DTD, the definition of their XMLs.

Worldpay's DTD: <http://dtd.worldpay.com/v1/>

What's a DTD: https://en.wikipedia.org/wiki/Document_type_definition

Under `y-ext/ext-worldpay/worldpayapi/resources/dtd` you can find 2 files, the original from Worldpay, so we can track their changes: `paymentService_v1_wp_original.dtd` and a java compatible one with the name `paymentService_v1.dtd`. This file is used in the buildcallbacks in `y-ext/ext-worldpay/worldpayapi/buildcallbacks.xml` to generate the classes to handle the XMLs from its definition.

If there is a change in the DTD, a new XSD file needs to be generated. The XSD file is used to validate the object representing the XML request to Worldpay.

This can be achieved by running `ant -f schemagen-build.xml`. This task will generate a file named "paymentService_v1.xsd" under `resources/schema`.

Reference: [https://en.wikipedia.org/wiki/XML_Schema_\(W3C\)](https://en.wikipedia.org/wiki/XML_Schema_(W3C))

Supported Environments

[Third-party compatibility](#)

Limitations

The Worldpay Payment Service API has different field length restrictions to the Hybris Accelerator / Commerce Suite. Email for example has a maximum length of 30 characters but Hybris accepts 255. The AddOn does not add any extra logic to check the length of these fields and this is expected to be customised on a project-by-project basis.

Languages

The provided AddOn only provides localization in English. The translations for other languages can be added by the integrator in the corresponding properties files on a project basis.

SAP Commerce 2105 Connector R4.0 - B2B Technical Implementation Guide

- [Reader's guide](#)
- [B2B Technical Implementation Guide](#)
 - [New B2B Accelerator Checkout Flow](#)
 - [Card Payment flow](#)
 - [Handling of 3D Secure](#)
 - [Requesting Security Code \(CVV\) on card payment](#)
 - [Business processes](#)
 - [Replenishment business flow](#)

Reader's guide

This technical implementation guide will cover the use of Worldpay as a payment provider in a B2B context.

B2B Technical Implementation Guide

The Worldpay B2B AddOn supports a payment scheme using Worldpay client-side encryption (CSE) and tokenizing of cards.

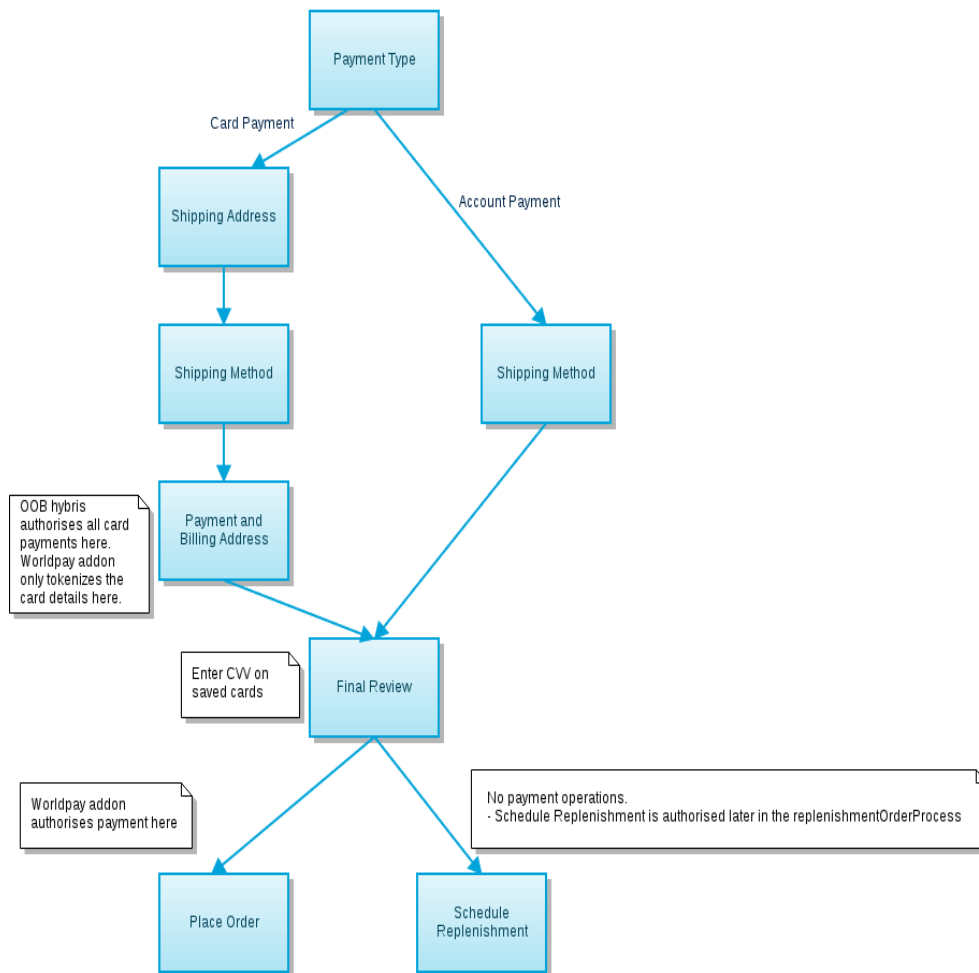
New B2B Accelerator Checkout Flow

The Worldpay B2B AddOn implements some changes to the OOB Hybris B2B checkout flow, changes in business processes in handling orders and finally additional request for the security code on card payment.

For reference see the OOB B2B checkout and order process is described here: [B2B Checkout and Order Process](#)

Card Payment flow

The figure below shows the different steps on the checkout page in the responsive UI experience. It is possible to checkout using card or account payment. The AddOn only changes the semantic of the card payment flow, the account payment is left untouched.



As shown in the figure above one of the main difference is when to execute payment operations. In the OOB solution, all card payments are initiated with an authorisation after the card details are entered. This is also the case for replenishment orders that are scheduled for later. The AddOn handles this by moving the authorisation of the payment transaction until the order scheme is known and selected after the final review of the order.

- If the customer is buying now by placing the order then the payment transaction is authorised.
- If the customer schedules a replenishment, the authorisation is handled by the generated Hybris cronjob, that is triggered according to the schedule. This authorisation is handled by a separate merchant configuration.

Handling of 3D Secure

The Worldpay B2B AddOn handles 3D secure in the B2B card payment flows. The supported scenario is:

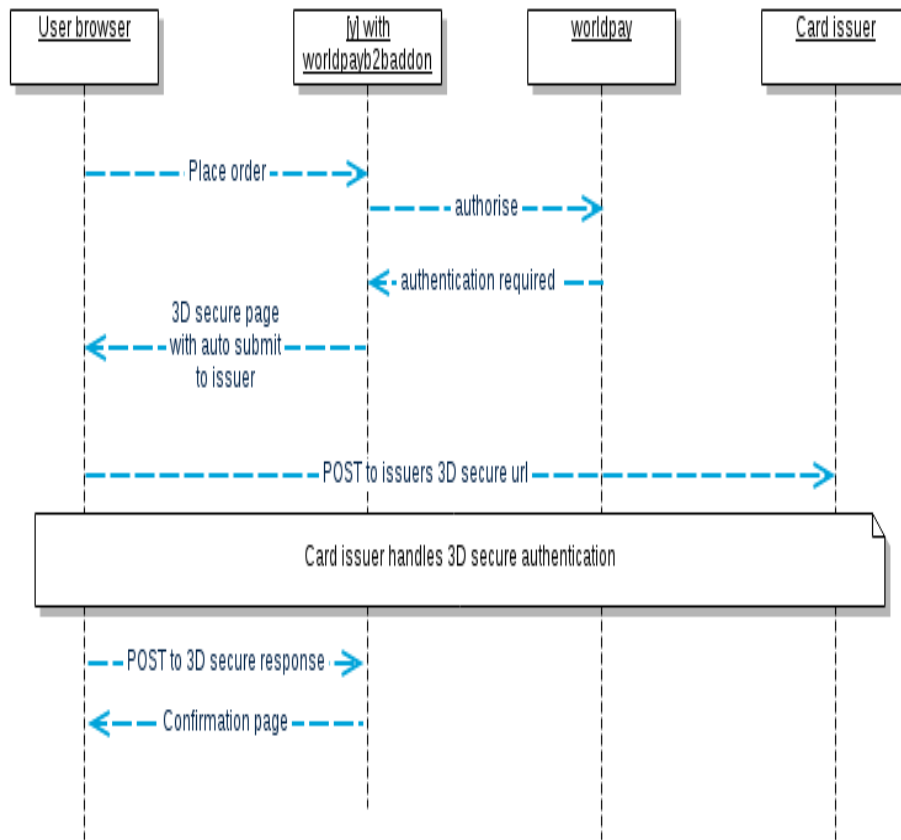
1. Place an order immediately from the order review step on the checkout page. The "buy now" scenario.

The unsupported scenario is:

1. A scheduled replenishment order created from a cronjob (no customer is present here for the 3D secure challenge).

The sequence diagram below shows the interactions between the customer's browser, a Hybris installation with the Worldpay B2B AddOn installed, Worldpay and the card issuer.

The 3D secure flow is started in the WorldpaySummeryCheckoutController /placeOrder endpoint in the buy now scenario. The result is a redirect to a page that auto submits a form to the card issuer. This form is implemented in the jsp page autoSubmit3DSecure.jsp. Besides the data needed by the card issuer, the form contains a return URL where the flow will continue after 3D secure.



Requesting Security Code (CVV) on card payment

The Worldpay B2B AddOn requests security code from the customer in the following scenarios on credit card payment:

1. When entering card details.
2. When using a saved card the security code must always be entered on the final review step on the checkout page. This happens before the customer decides between: placing the order now or scheduling a replenishment order.

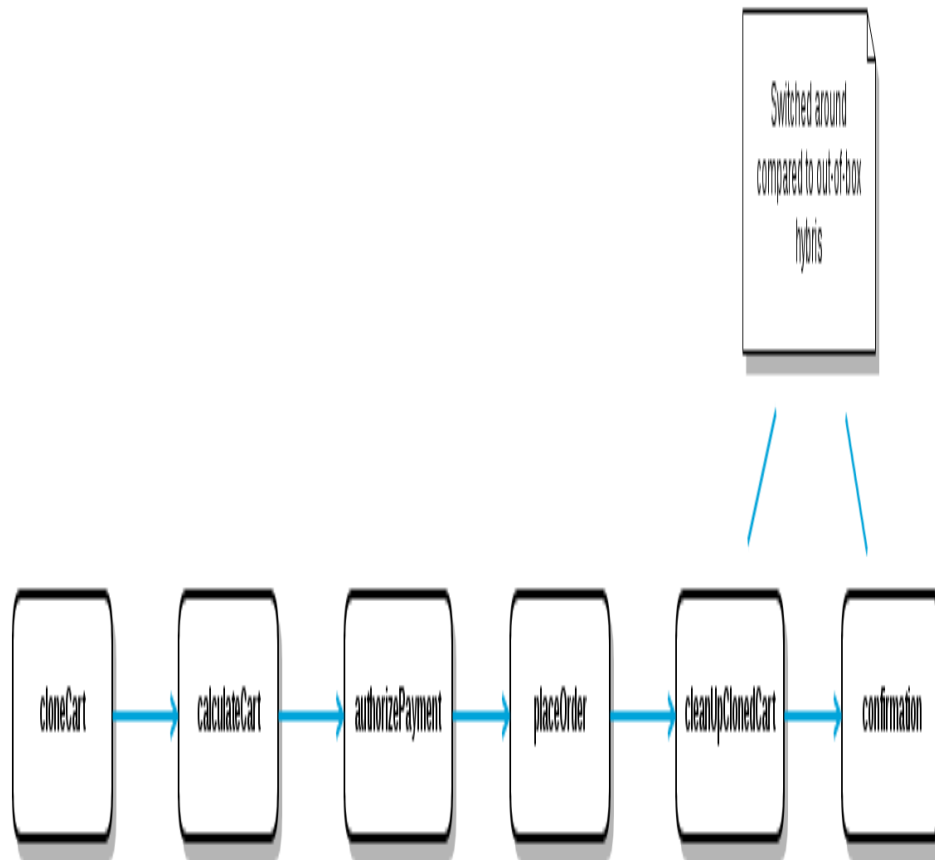
There is no customer present when replenishment orders are authorised within the generated cronjob, hence it is not possible to supply a security code in this scenario.

There is no validation of the security code within the AddOn, this is handled entirely by Worldpay.

Business processes

Replenishment business flow

When the "CartToOrderCronJob" runs in order to generate a new Order from the replenishment Cart, it triggers a business process: replenishmentOrderProcess.xml. In the Worldpay B2B AddOn this process has been customized in a worldpayReplenishmentOrderProcess.xml in order to fix a known concurrency issue in the OOB process when including the assistedServiceFacades extension in the setup. The only change in the process flow is to remove the newly cloned Cart before executing the confirmationAction which triggers a new business process to send a confirmation mail to the customer.



Two replenishment actions have been customized in the Worldpay B2B AddOn:

cloneCartAction - customized in WorldpayCloneCartAction:

The out-of-box CloneCartAction creates a new Cart from the cartToOrderCronJob and uses a single KeyGenerator to set both the code and guid of the newly created Cart. The Worldpay B2B AddOn uses the code of a Cart to generate a Worldpay order-code and for this to work, the code of the Cart needs to be on the normal orderCodeGenerator form. The customized WorldpayCloneCartAction enables specifying separate KeyGenerator for the code and guid.

authorizePaymentAction - customized in WorldpayAuthorizePaymentAction:

The WorldpayAuthorizePaymentAction uses the WorldpayDirectOrderFacade to authorize the payment against Worldpay. The authorize is done using a special replenishment Merchant (see [SAP Commerce 2011 Connector R1.0 - Implementation Guide](#) for configuration)