SAP Commerce 2205 Connector R1.1 - 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 worldpayextocc 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 worldpayextocc 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 hybris OCC interface see:

Related Documentation

- OCC API documentation, version v2

Endpoints on the WorldpayCartsController

Method	Path	Parameters
POST	/users/{userId}/carts/{cartId}/worldpaypaymentdetails	The hybris OOB PaymentDetails is extended with a cseToken

Endpoints on the WorldpayOrdersController

Method	Path	Parameters
POST	/users/{userId}/worldpayorders	• cartId - the id of the used shopping cart • securityCode - the security code for the used credit card
POST	/users/{userId}/worldpayorders/3dresponse	• 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	• cartId - the id of the used shopping cart • challengeWindowsSize - size of the modal used for 3ds challenge (if required) ◦ supported: 250x400, 390x400, 500x600, 600x400 • dReferenceId - reference id given during DDT • securityCode - CVC of the payment info
POST	/users/{userId}/worldpayorders/3dresponse	• cartId - the id of the used shopping cart • paRest - the 3D protocols payer authentication response • merchantData: Worldpay order code

Endpoints on WorldpayOccApi

Method	Path	Parameters
GET	/{baseSiteId}/worldpayapi/cse-public-key	n/a
GET	/{baseSiteId}/worldpayapi/ddc-3ds-jwt	n/a
POST	/{baseSiteId}/worldpayapi/challange/submit	• TransactionId - transaction id sent by 3ds challenge (optional) • Response - arbitrary string (optional) • MD - worldpay order code

Endpoints on the GooglePay

Method	Path	Parameters	Reference links
POST	/checkout/multi/worldpay/googlepay/authorise-order	token - protocolVersion - The protocol Version - signature - The signature - signedMessage - The signed message billingAddress - 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 - sortingCode - The shorting code	*https://developers.google.com/pay/api/web/guides/resources/payment-data-cryptography*
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	Token - ApplePayPaymentMethod -- displayName -- network -- type -- paymentPass --- primaryAccountIdentifier --- primaryAccountNumberSuffix --- deviceAccountIdentifier --- deviceAccountNumberSuffix --- activationState - transactionIdentifier - paymentData -- header --- ephemeralPublicKey --- publicKeyHash --- transactionId -- signature -- version -- data - billingContact -- 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	• validationURL: Apple API endpoint for requiring payment session	

POST	/{baseSiteId}/users/{userId}/carts/{cartId}/apple/authorise-order	• Token ◦ ApplePayPaymentMethod -- displayName -- network -- type -- paymentPass --- primaryAccountIdentifier --- primaryAccountNumberSuffix --- deviceAccountIdentifier --- deviceAccountNumberSuffix --- activationState ◦ transactionIdentifier ◦ paymentData -- header --- ephemeralPublicKey --- publicKeyHash --- transactionId -- signature -- version -- data ◦ billingContact -- phoneNumber -- emailAddress -- givenName -- familyName -- phoneticGivenName -- phoneticFamilyName -- addressLines[ ] -- subLocality -- locality -- postalCode -- subAdministrativeArea -- administrativeArea -- country -- countryCode ◦ shippingContact: Same fields as billingContact	

Payment flows

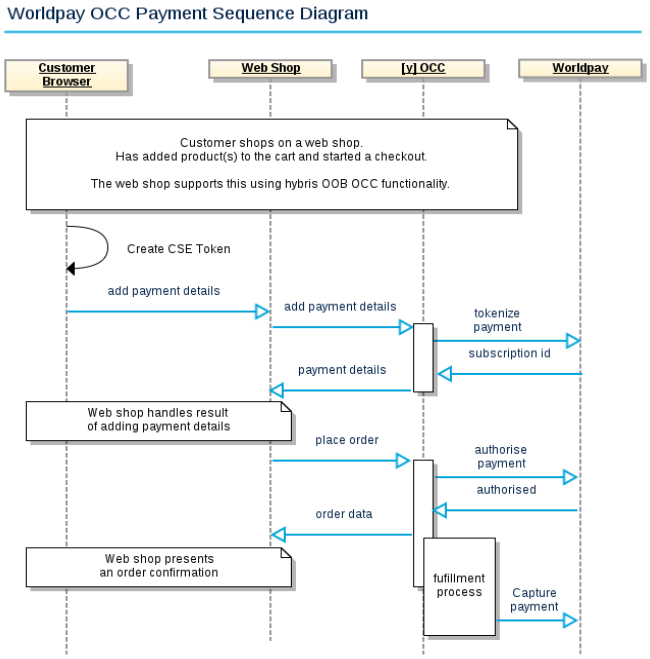
The section gives an overview of the Worldpay payment flows.

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

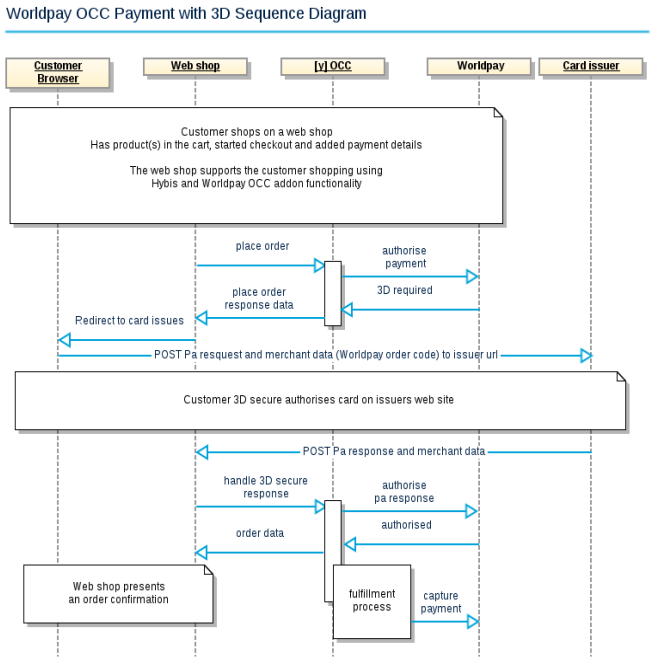
Related Documentation

- Customer Buying Process Scenarios

Payments flow without 3D secure

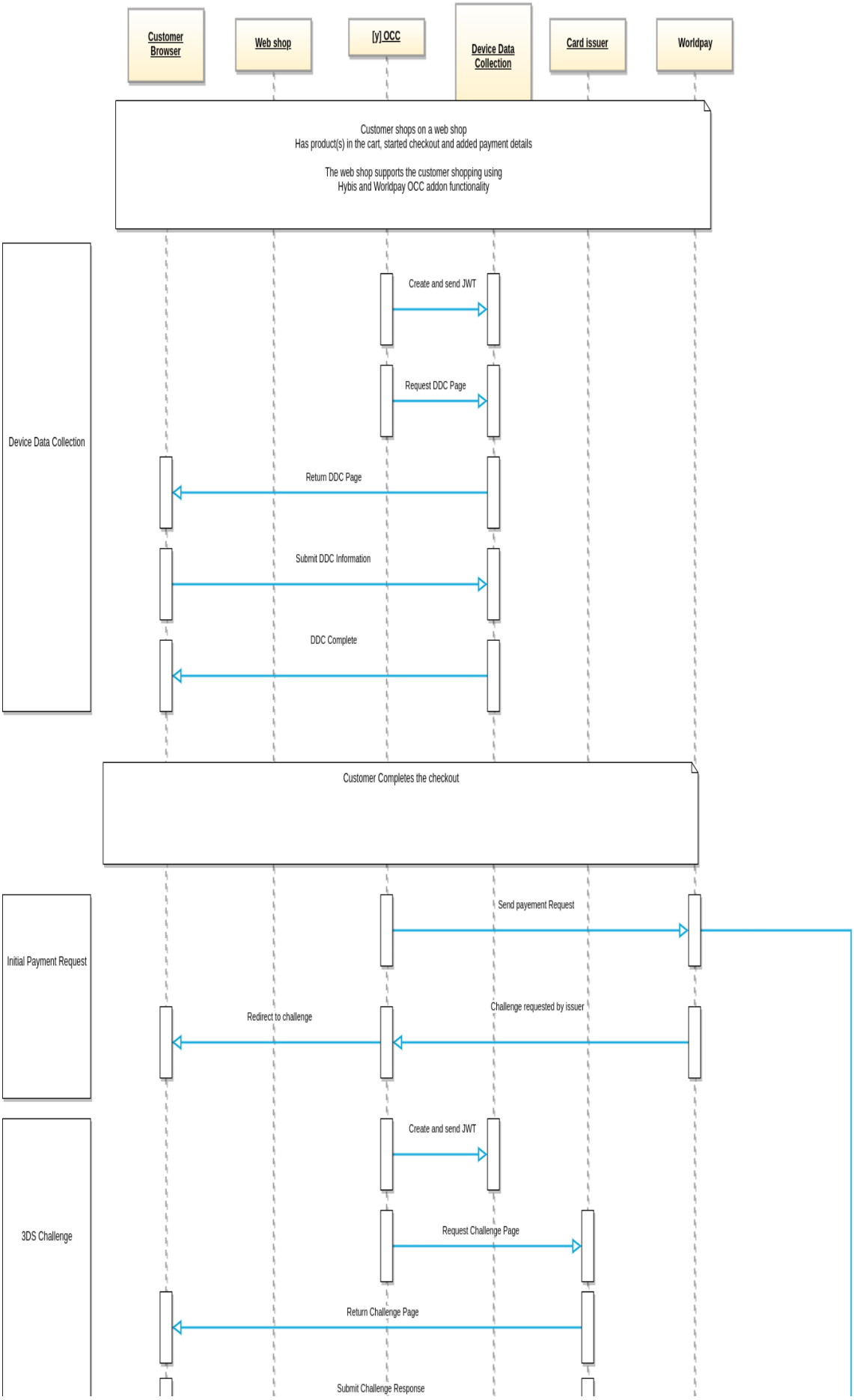


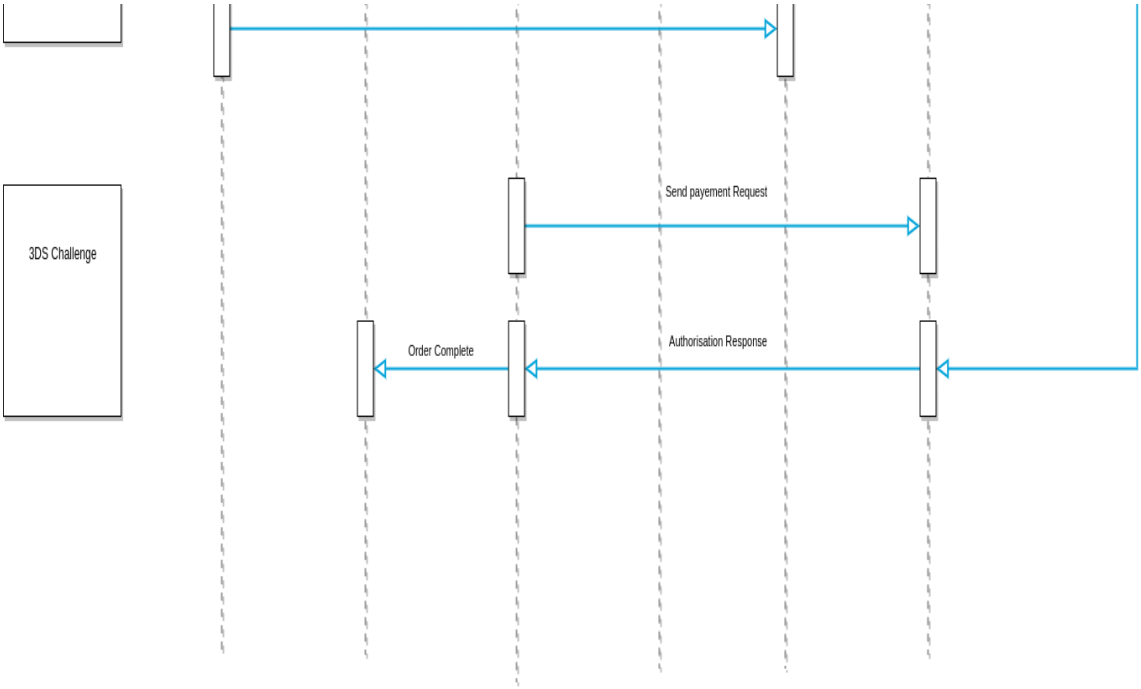
Payments flow with 3D secure



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 paged 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 worldpayresponcemock 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\:/
```

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