worldpay

Worldpay Testing Guide

Version 19.2.0

1. SUMMARY

This document provides Test cases for each Worldpay service integrated within SFCC platform with MobileFirst Reference Application version to ensure main functionality is operating correctly.It includes Automation Test Suite for Functional Testing, Integration Testing and Unit Testing to verify the implementation.

It also includes Codecept Testing Automation Suite for Acceptance Testing.

1.1 TEST CASES

This Cartridge has gone through QA and UAT including live proving across all the payment methods. Please refer to the test cases attached here.

Worldpay_TestCase
s_summaryPhase2.xl

1.2 TEST DATA

Refer to Appendix A for values of test card numbers.

1.3 Automation test suit for functional testing

Pre Requiste

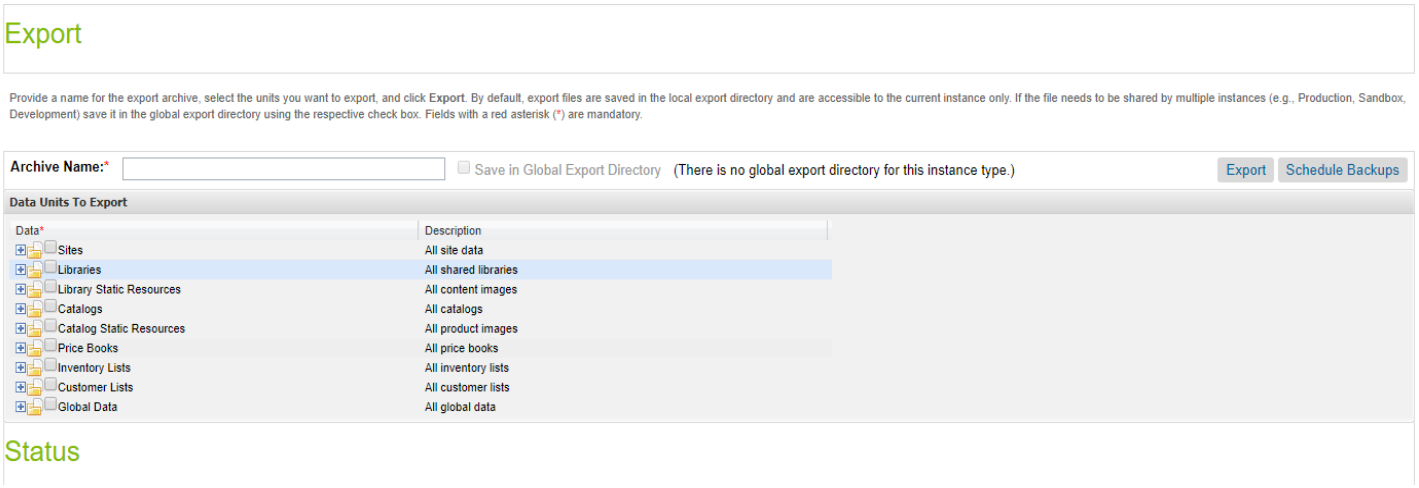
1. Redirect credit card payment should be in page format so remove any config did for iframe or light box in payment method.

2. We chat redirect and direct payment can be test one at a time. So while testing redirect mode make the line change around 22(describe =>describe.skip) in wechatDirectPaymentMethod.js and vice versa on the file wechatRedirectPaymentMethod.js while testing the we chat direct mode.

3. Order confirmation page customer zipcode, customer country and customer phone number comparison can be done with the data configured in test/functional/mocks/customers.js. So we recommend to create one new customer for automation testing which having the address zip code '14304' and country code 'US' and phone number '3333333333'. Save Visa card '4917610000000000' for that customer.

4. Do Site export, the required xml files are: customer-list, catalogs, pricebooks, promotions

In Business Manager, Administration => Site Development => Site Import and Export



Select which one you want to take export of like Price Books, select all price books, give

Archive Name and click on export after the success status you can download that file/folder by clicking on download button.

5. Update the folder path and file name to be sync with the path given in the file functional/mocks/testDataMgr/main.js.

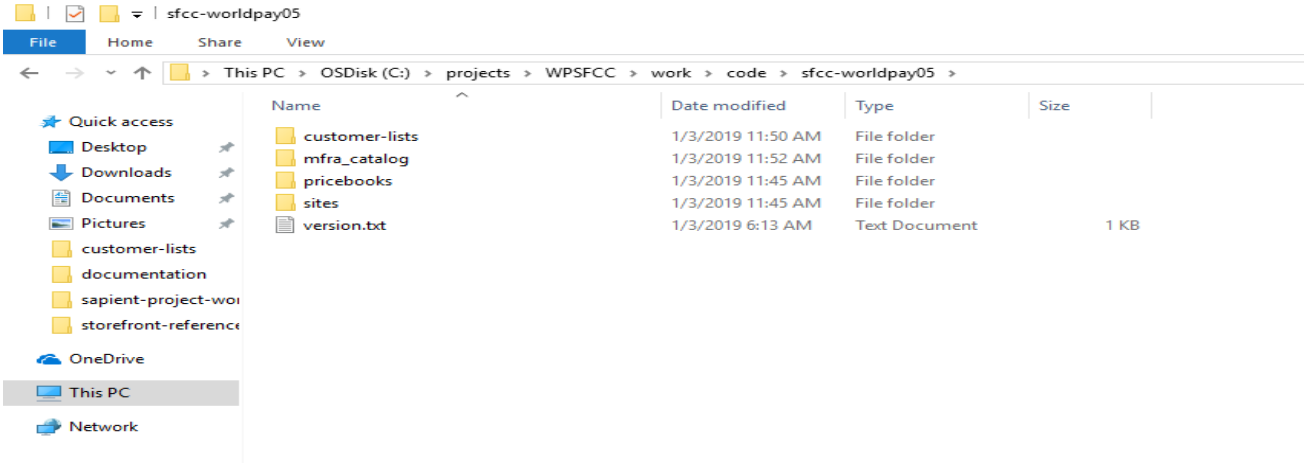
a) For example
Customer xml path is mentioned in the file like this

```
files: [demoDataDir + '/customer-lists/customer.xml']
```

The same path should exist in the site export directory, if you site export direct contains the file path like this C:\projects\WPSFCC\work\code\sfcc_worldpay\customer-lists\{siteid}.xml means you need to rename the file name and make the structure C:\projects\WPSFCC\work\code\sfcc_worldpay\customer-lists\customer.xml like this.

So check the xml folder structure corresponding the path mentioned in the file.

Site export folder structure should like this



b. There are some issues in parsing customer-list, so in the customer lists customer xml file update the xml string 'customer-list' to 'customers'. Or you can export all the customer directly from business manager and replace that xml in the 'customer-lists' folder.

6. In worldpay_main.js, functional/mocks/testDataMgr/worldpay_main, provide the required credentials which are existing in the site

a) loginUserEmail – existing user email(recommend user email in 3rd point)

b) loginPassword - existing user password

c) newUserPassword – new user registration password

d) storefrontPath – absolute folder path to the site export

e) resourcePath – absolute folder path to storefront reference architecture resource folder

f) 3 Variant Product ID like variantProduct1, variantProduct2, variantProduct3(example product ids which are having good inventory)

g) All credit card details scenario you want to check like creditCardAmex, creditCardDiscover, creditCardVisa, creditCardError, creditCardRefused, creditCard3D containing card name, card type, number, expiry date, cvv number.

Note:- Login user email should be available in the customer list xml file also.

7. If dw.json is not there in root folder, create one containing following :

```
{
  "hostname": "worldpay03-tech-prtnr-eu04-dw.demandware.net",
  "username": "USERNAME_OF_STOREFRONT",
```

```
  "password": "PASSOWRD_OF_STOREFRONT",

  "code-version": "CODE_VERSION"

}
```

Steps to configure and run the automation testing

1. Run command npm install in root folder
2. Download selenium-standalone globally and start selenium server using command:-

npm install selenium-standalone@latest -g

selenium-standalone install

selenium-standalone start

3. To run functional testing, run command “selenium-standalone start” in one terminal and the command to run test “npm run test:functional” in another terminal. Functional automation testing will be started running each file individually in the folder ‘test/functional/checkout’

The test scenarios are,

- For Guest User :-
  Direct Credit Card Payment Method– directcreditcardPaymentMethod.js

  Redirect Credit Card Payment Method – redirectcreditcardPaymentMethod.js

  Wechat Direct Payment Method – wechatDirectPaymentMethod.js

  Wechat Redirect Payment Method – wechatRedirectPaymentMethod.js

  Paypal Payment Method – paypalPaymentMethod.js

  Refused Credit Card Payment Method – refusedCreditCardPaymentMethod.js

- For Registered User :-
  Login and using Saved cards (User account should have saved cards for this)

  Register Customer and Direct Credit Card Payment Method

  Sepa Payment Method – sepaPaymentMethod.js

  Alipay Payment Method – alipayPaymentMethod.js

  3D Payment Method – 3dPaymentMethod.js

  Mister cash Payment Method –mistercashPaymentMethod.js

Error Credit Card Payment Method– errorCreditCardPaymentMethod.js

To run single scenario/single file only change the specs line(around 88) in the file 'test/functional/webdriver/wdio.conf.js'. For ex. To test only the savecard functionality change the line

```
specs: [
specs
  ],
```

To

```
specs: [
'test/functional/checkout/registeredUser/loginSavedCard.js'
  ],
```

Note:- You should be able to see count of testcases passing, failing and skipped ones. And also required order number in the console.

Integration Testing

1. Open terminal and run command "npm run test:integration" to run the test/integration files one by one and you will be able to see the required response using different payment options and for failed scenario, only error message is printed.
   Note:- The response we get is of "CheckoutServices-PlaceOrder".

2. The test scenarios are as follows:-

   - 3DCreditCardPaymentMethod
   - alipayPaymentMethod
   - errorCreditCardPaymentMethod
   - loginDirectCreditCardPaymentMethod
   - loginSavedCardPaymentMethod
   - mistercashPaymentMethod
   - paypalPaymentMethod
   - redirectCreditCardPaymentMethod
   - sepaPaymentMethod
   - wechatPaymentMethod

For integration testing :-

https://medium.com/adobetech/how-to-combine-rest-api-calls-with-javascript-promises-in-node-js-or-openwhisk-d96cbc10f299

1.4 Automation test suit (Codeceptjs)

Steps to configure and run the Automated Acceptance Testing on Codecept

- *Add codeceptjs and update selenium-standalone dependencies in **package.json** to 5.8.0 and above version to run Acceptance Testing on Codecept*
  "@wdio/selenium-standalone-service": "^5.8.0",
  "codeceptjs": "^2.1.0"

  *Note: In order to run the Functional Testing on chai mocha,add chai dependencies and update the version of selenium-standalone dependencies in **package.json***

  "wdio-selenium-standalone-service": "0.0.11",

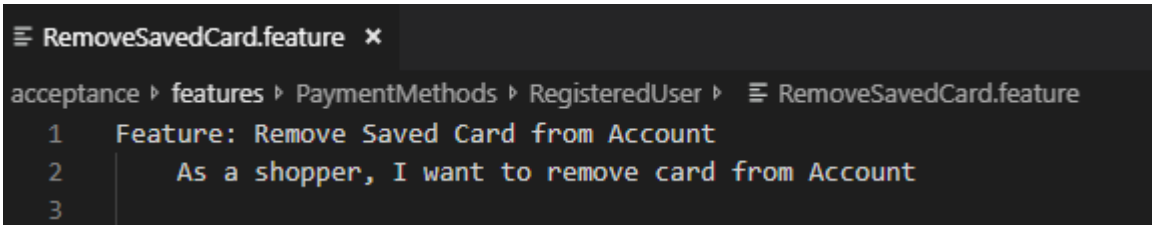
  Run npm install everytime the version is changed.

- To run Acceptance testing, run command "**selenium-standalone start**" in one terminal and the command to run test "**npm run test:acceptance**" in another terminal.
  CodeceptJS configuration is set in **codecept.conf.js file**.This is how the configuration file looks like:

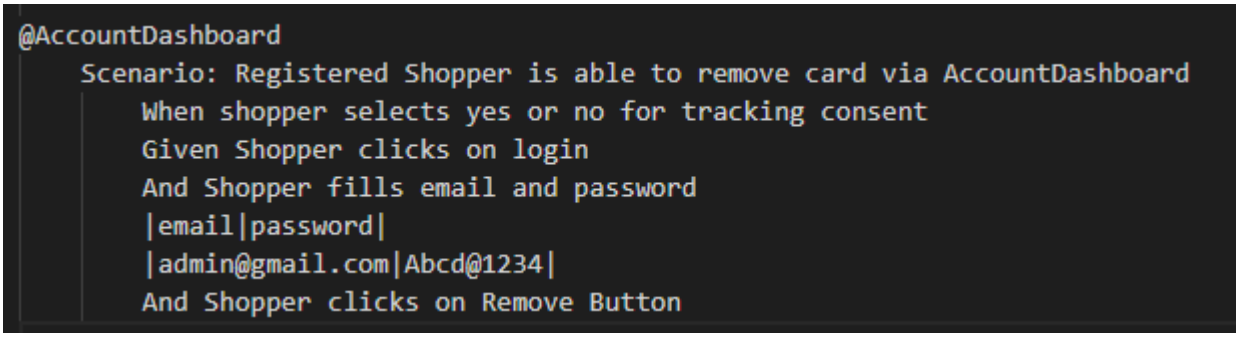
```
var RELATIVE_PATH = './test/acceptance';
var OUTPUT_PATH = RELATIVE_PATH + '/report';
var HOST = 'https://worldpay01-tech-prtnr-eu04-dw.demandware.net';

varwebDriver = {
   url: HOST,
   browser: 'chrome',
smartWait: 10000,
waitForTimeout: 10000,
   timeouts: {
      script: 60000,
      'page load': 10000
   }
};

exports.config = {
   output: OUTPUT_PATH,
   helpers: {
      WebDriver: webDriver
   },
   plugins: {
wdio: {
         enabled: true,
         services: ['selenium-standalone']
      },
      allure: {
```

```
        enabled: true
      },
retryFailedStep: {
        enabled: true,
        retries: 1
      }
   },
   include: {
homePage: RELATIVE_PATH + '/pages/HomePage.js',
                        worldpayPaymentTestRegistered:  RELATIVE_PATH + '/pages/worldpayPaymentTestRegistered.js',
uriUtils: RELATIVE_PATH + '/utils/uriUtils.js'
   },


gherkin: {
      features: RELATIVE_PATH + '/features/PaymentMethods/**/IdealRegistered.feature',

      steps: [
         RELATIVE_PATH + '/features/steps/land_home_page.steps.js',
                                 RELATIVE_PATH + '/features/steps/idealRegistered.steps.js',
      ]
   },
   tests: RELATIVE_PATH + '/tests/**/*.test.js',
   name: 'link_worldpay'
};
```

Features *-A feature would describe the current test script to be executed.Every *.feature file conventionally consists of a single feature. Lines starting with the keyword* ***Feature:****starts a feature. A feature usually contains a list of scenarios.*
Eg:RemoveSavedCard.feature



Scenarios *-Scenario describes the steps and expected outcome for a particular test case.Every scenario starts with the Scenario: keyword. Each feature can have one or more scenarios, and every scenario consists of one or more steps.*



Steps -*Features consist of steps, also known as* Givens*,* Whens *and* Thens*.*

- ***Given****: It specifies the context of the text to be executed.*
- ***When****: "When" specifies the test action that has to performed*
- ***Then****: The expected outcome of the test can be represented by "Then"*

Testing will be started with running the features mentioned in **gherkin** which will then look for the steps to be executed.

```
include: {
    homePage: RELATIVE_PATH + '/pages/HomePage.js',
    worldpayPaymentTestRegistered:  RELATIVE_PATH + '/pages/worldpayPaymentTestRegistered.js',
    uriUtils: RELATIVE_PATH + '/utils/uriUtils.js'
},
gherkin: {
    features: RELATIVE_PATH + '/features/PaymentMethods/**/IdealRegistered.feature',

    steps: [
        RELATIVE_PATH + '/features/steps/land_home_page.steps.js',
        RELATIVE_PATH + '/features/steps/idealRegistered.steps.js',
    ]
```

3. The test scenarios are as follows:-

- **Guest User**

  Create User

  Direct Credit Card Payment(Master, Visa, Amex, 3ds1, 3ds2, Refused/Cancelled)

  Redirect Credit CardPayment Method

  PaypalPayment Method

  AlipayPayment Method

  SofortPayment Method

  IdealPayment Method

  SepaPayment Method

  GooglePayPayment Method

  WechatPayPayment Method

- **Registered User**

Remove Saved Card from My Account

Direct Credit CardPayment Method (Master, Visa, Amex, 3ds1, 3ds2, Refused/Cancelled)

Redirect Credit Card Payment Method

PaypalPayment Method

AlipayPayment Method

SofortPayment Method

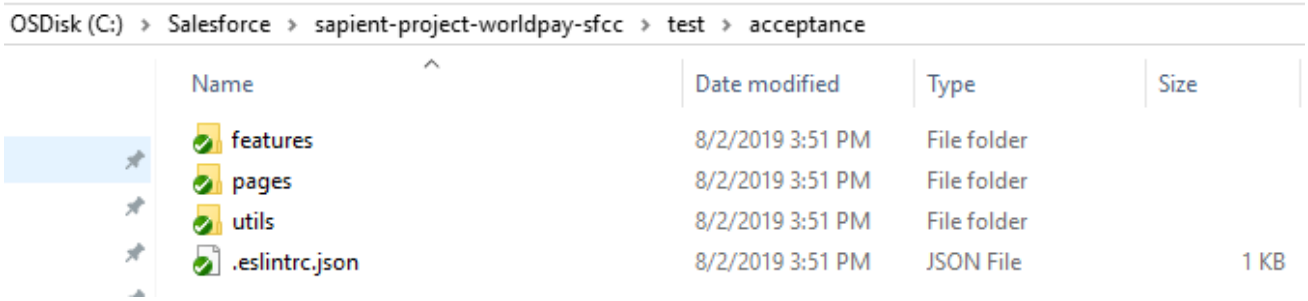
IdealPayment Method

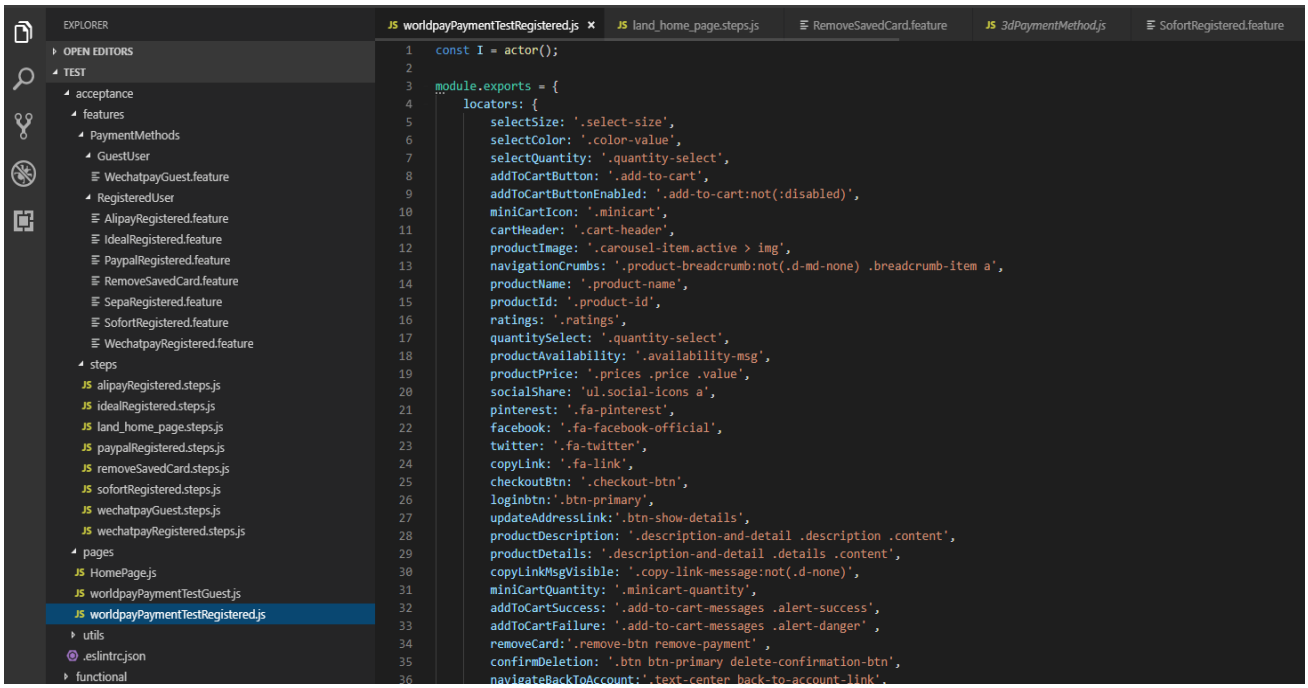
Sepa Payment Method

GooglePayPayment Method

WechatPayPayment Method

Save Credit card (Direct) and place order through saved card





Unit Testing

Open terminal and run command “npm run test” to run the test/unit files

APPENDIX **A: APM/Card mapping keys**

Supported Cards Mapping table:

Card Name	Key Value	Test card number
Airplus	AIRPLUS-SSL	122000000000003
American Express	AMEX-SSL	34343434343434
Dankort	DANKORT-SSL	5019717010103742
Diners	DINERS-SSL	36700102000000
Discover card	DISCOVER-SSL	6011000400000000
JCB	JCB-SSL	3528000700000000
Laser	LASER-SSL	630495060000000000 630490017740292441
Maestro	MAESTRO-SSL	6759649826438453, 6799990100000000019
MasterCard	ECMC-SSL	5555555555554444, 5454545454545454
Visa	VISA-SSL	4444333322221111, 4911830000000

Supported Payment Method Mapping table:

Payment Method Name	Key Value
Konbini	KONBINI-SSL
Poli	POLI-SSL
Poli NZ	POLINZ-SSL
Przelewy24	PRZELEWY-SSL
SEPA-DD	ELV-SSL
Alipay	ALIPAY-SSL
Boleto	BOLETO-SSL
CashU	CASHU-SSL
Credit Card – Direct	CREDIT_CARD
Credit Card – Redirect	Worldpay
China Union Pay	CHINAUNIONPAY-SSL
ENETS	ENETS-SSL
Giropay	GIROPAY-SSL
IDEAL	IDEAL-SSL
Mistercash	MISTERCASH-SSL
Paypal	PAYPAL-EXPRESS
Sofort	SOFORT-SSL
Qiwi	QIWI-SSL
Yandex	YANDEXMONEY-SSL

APPENDIX B: IDEAL Bank List

Bank Name	Bank Code
ABN	ABN_AMRO
ASN	ASN
ING	ING
Knab	KNAB
Rabobank	RABOBANK
SNS	SNS
SNS Regio	SNS_REGIO
Triodos	TRIODOS
Van Lanschot	VAN_LANSCHOT

APPENDIX C: Error Codes and Error Messages for Transaction Notification

Master Error Code	Error Message
111	XML Parse error has occurred
112	XML Corrupted, Could not find Order No.
113	XML in Custom Object Corrupted
114	Error occurred while deleting Custom Object
115	Error occurred while reading status history from Order Object
116	Error occurred while Updating Order Object
117	Error occurred while reading Custom Object
118	No Transaction History Available
119	Last status Not available as No Transaction History is Available
120	Wrong Order Number

APPENDIX D: Order Notification and SFCC Order status mapping

Order Notifications Received From Worldpay	Changes in Order object in Business Manager
AUTHORISED	Order Status: NEW Payment Status: no change Export Status: READY FOR EXPORT Confirmation Status : CONFIRMED
CANCELLED	Order Status: FAILED/CANCELLED Payment Status: NOT PAID Export Status: NOT EXPORTED Confirmation Status : NOT CONFIRMED
CAPTURED	Order Status: COMPLETED Payment Status: PAID Export Status: no change Confirmation Status : no change
SENT_FOR _REFUND	Order Status: no change Payment Status: no change Export Status: no change Confirmation Status : no change
REFUSED	Order Status: FAILED Payment Status: NOT PAID Export Status: NOT EXPORTED Confirmation Status : NOT CONFIRMED
SETTLED	Order Status: no change Payment Status: no change Export Status: no change Confirmation Status : no change
INFORMATION_REQUESTED	Order Status: no change Payment Status: no change Export Status: no change Confirmation Status : no change
CHARGED_BACK	Order Status: no change Payment Status: no change Export Status: no change Confirmation Status : no change
EXPIRED	Order Status: FAILED Payment Status: NOT PAID Export Status: NOT EXPORTED Confirmation Status : NOT CONFIRMED

APPENDIX E: Error Codes and Error Messages

Master Error Code	Error Message
worldpay.error.code1	Internal error has occurred , please choose a different Payment Method or try again later.
worldpay.error.code2	Parse error has occurred, please choose a different Payment Method or try again later.
worldpay.error.code3	Invalid amount error, please choose a different Payment Method or try again later.
worldpay.error.code4	Security error , please choose a different Payment Method or try again later.
worldpay.error.code5	Invalid request, please choose a different Payment Method or try again later.
worldpay.error.code6	Please choose a different Payment Method or try again later.
worldpay.error.code7	Payment details in the order element are incorrect, please choose a different Payment Method or try again later.
worldpay.error.code8	Submission error , please choose a different Payment Method or try again later.
worldpay.error.generalerror	Please choose a different Payment Method or try again later.
worldpay.error.cancelerror	Your transaction has been cancelled.